(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 September 2002 (26.09.2002)

PCT

(10) International Publication Number
WO 02/075547 A1

(51) International Patent Classification[7]: G06F 11/30

(21) International Application Number: PCT/US02/08029

(22) International Filing Date: 15 March 2002 (15.03.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/809,030 16 March 2001 (16.03.2001) US

(71) Applicant *(for all designated States except US)*: KAVADO, INC. [US/US]; 421 West 118th Street, Suite 51, New York, NY 10027 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: BEN-ITZHAK, Yuval [IL/IL]; Shtern St. 92/a, Kiron 55602 (IL).

(74) Agents: CODDINGTON, Trevor, Q. et al.; Intellectual Property Department, Brobeck, Phleger & Harrison LLP, 1333 H Street, N.W., Suite 800, Washington, DC (US).

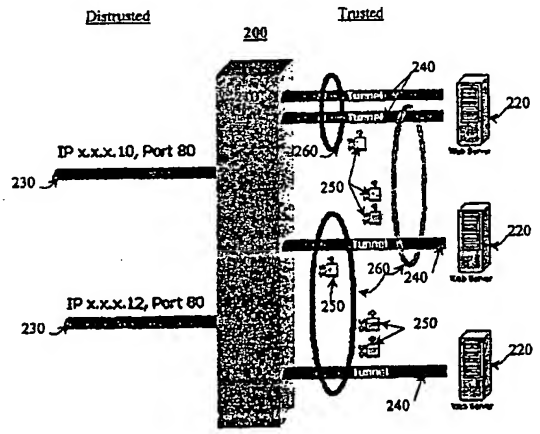
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: APPLICATION LAYER SECURITY METHOD AND SYSTEM



(57) Abstract: Figure 2A illustrates an application layer security system (200) according to an embodiment of the invention. Particularly, protective layer (210) is implemented between a distrusted environment and trusted environment including one or more applications residing on one or more web servers (220). Protective layer (210) protects trusted applications from maliciious operation requests sent via a distruted source. Each incoming operation request can comprise a plurality of requested commands, parameters, and parameter values to be performed by and application. Incoming operation requests may be received as external HTTP traffic via one or more ports (230). Security system (200) includes tunnels (240) to support single, distributed, or share applications. Tunnels (240) are protocol based, e.g., TCP/IP, connections between the distrusted and trusted environments. Each tunnel (240) can implement and validate a specified application protocol, such as, for example, HTTP, HTTPs, SOAP. System (200) further includes one or more pipes (250) to provided to protect or monitor functionality of the applications, and the system also includes web applications (260). Further, web applications (260) can customize pipes execution to specific application paths.

WO 02/075547 A1

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 September 2002 (26.09.2002)

PCT

(10) International Publication Number
WO 02/075547 A1

(51) International Patent Classification[7]: G06F 11/30

(21) International Application Number: PCT/US02/08029

(22) International Filing Date: 15 March 2002 (15.03.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/809,030 16 March 2001 (16.03.2001) US

(71) Applicant *(for all designated States except US)*: **KAVADO, INC.** [US/US]; 421 West 118th Street, Suite 51, New York, NY 10027 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **BEN-ITZHAK, Yuval** [IL/IL]; Shtern St. 92/a, Kiron 55602 (IL).

(74) Agents: **CODDINGTON, Trevor, Q.** et al.; Intellectual Property Department, Brobeck, Phleger & Harrison LLP, 1333 H Street, N.W., Suite 800, Washington, DC (US).

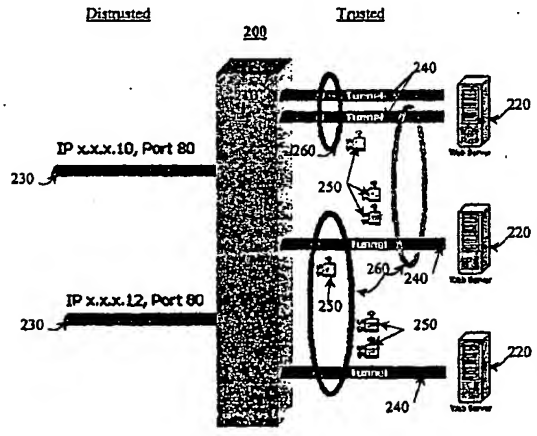
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: APPLICATION LAYER SECURITY METHOD AND SYSTEM



(57) Abstract: Figure 2A illustrates an application layer security system (200) according to an embodiment of the invention. Particularly, protective layer (210) is implemented between a distrusted environment and trusted environment including one or more applications residing on one or more web servers (220). Protective layer (210) protects trusted applications from maliciious operation requests sent via a distruted source. Each incoming operation request can comprise a plurality of requested commands, parameters, and parameter values to be performed by and application. Incoming operation requests may be received as external HTTP traffic via one or more ports (230). Security system (200) includes tunnels (240) to support single, distributed, or share applications. Tunnels (240) are protocol based, e.g., TCP/IP, connections between the distrusted and trusted environments. Each tunnel (240) can implement and validate a specified application protocol, such as, for example, HTTP, HTTPs, SOAP. System (200) further includes one or more pipes (250) to provided to protect or monitor functionality of the applications, and the system also includes web applications (260). Further, web applications (260) can customize pipes execution to specific application paths.

WO 02/075547 A1

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

APPLICATION LAYER SECURITY METHOD AND SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to information security, particularly to an application layer security method and system for protecting trusted computer applications from executing illegal or harmful operation requests originating from an unknown or distrusted source.

2. Description of Background

The ease, accessibility, and convenience of the Internet have rapidly changed the way people use computers and access information. The World Wide Web ("WWW"), often referred to as 'the web,' is one of the most popular means for retrieving information on the Internet. The web gives users access to an almost infinite number of resources such as interlinked hypertext documents retrieved via a hypertext transfer protocol ("HTTP") from servers located around on the world. The web operates in a basic client-server format, wherein servers are dedicated computers or individual computer applications that execute resources in a certain matter, such as storing and transmitting web documents or binary objects, to client computers on the network. For example, a user can interact with a server through a web browser in order to view retrieved information or to request an application on the server to operate in a desired manner.

Documents on the web, referred to as web pages, are typically written in a hypertext markup language ("HTML") or similar language, and identified by uniform resource locators ("URLs") that specify a particular machine and pathname by which a file or resource can be accessed. Codes, often referred to as tags, embedded in an HTML document associate particular words and images in the document with URLs so that a user can access another file or page by pressing a key or clicking a mouse. These files generally comprise text, images, videos, and audio, as well as applets or other embedded software programs, written in for example, Java or ActiveX, that execute when the user activates them by clicking on a hyperlink. A user viewing a web page may also interact with components that, for example, forward requested information supplied by the user to a server through the use of forms, download files via file transfer

protocol ("FTP"), facilitate user participation in chat rooms, conduct secure business transactions, and send messages to other users via e-mail by using links on the web page.

Typically, the components that legitimate users desire and that are likely to make a web site spectacular or popular can also make a server and surrounding network environment vulnerable to attack from malicious, irresponsible, or criminally-minded individuals. This is referred to as "web hacking" and generally involves taking advantage of mistakes or vulnerabilities in web design through the server applications themselves. Web hacking is different from traditional system or application hacking because an attack generally takes place via application layer protocols. Generally, the easier it is for clients to talk or interact directly to the server applications through a web page, the easier it is for someone to hack into those applications. Typical attacks include, but are not limited to, defacing a page by deleting graphics and replacing them with doctored, sometimes lurid, graphics; altering or stealing password files; deleting data files; pirating copyrighted works; tampering with credit and debit card numbers, or other customer information; publicizing private business information; accessing confidential or unauthorized information; and searching through internal databases. Thus, web hacking causes inconvenience and perhaps irreversible damage to users, customers, businesses, and operators of the server(s). Generally, conventional computer security methods fail to address or completely ignore web hacking concerns.

The International Organization for Standardization ("ISO") developed a set of protocol standards designed to enable computers to connect with one another and to exchange information with as little error as possible. The protocols generally accepted for standardizing overall computer communications are designated in a seven-layer set of hardware and software guidelines known as the open systems interconnection ("OSI") model. This protocol model forms a valuable reference and defines much of the language used in data communications. As illustrated in **Fig. 1**, the application layer is the highest layer of standards in the OSI model.

Conventional security methods are typically implemented between either the data link layer and physical layer by using a firewall or the session and transport layers by using a secure socket layer ("SSL") or public key infrastructure ("PKI"). A

firewall is a type of security implementation intended to protect a trusted environment or network against external threats at the data link layer originating from another network, such as the Internet. A firewall prevents computers behind the firewall from communicating directly with computers external to the trusted network. Instead, all communications are routed through a proxy server outside of a trusted network, and the proxy server decides whether it is safe to let a particular message type or file type pass through, based on a set of filters, to the trusted network. A secure socket layer is an open standard developed by Netscape Communications® for establishing a secure and encrypted communications channel to prevent the interception of critical information, such as credit card information. The primary purpose of using SSL is to enable secure and encrypted electronic transactions on public networks, such as the web. A public key infrastructure or trust hierarchy is a system of digital certificates, certificate authorities, and other registration authorities that verify and authenticate each party involved in a communication session. PKIs are currently evolving and there is no single PKI nor even a single agreed-upon standard for setting up a PKI. One drawback of the above noted conventional technologies is that they do not perform an inspection of the application layer protocol, i.e., they do not scrutinize the application content of an incoming request. Therefore, these technologies can not prevent web hacking attacks directed through the application content of an operation request.

Web hackers can easily attack computer systems by exploiting flaws and vulnerabilities in web design. For example, default scripts may allow files to be uploaded onto a web server; a web server's treatment of environmental variables may be exploited; and the existences of 'backdoors' or flaws in third party products allow unauthorized access. These techniques can be potent attacks and are generally difficult to defend against through conventional means. Each month new software vulnerabilities are discovered, but many system operators typically leave these holes unpatched and their systems open to preventable attacks. Major corporations and government agencies utilizing well configured firewalls, PKI, and SSL implementations have been infiltrated by hackers using known application layer intrusions. These intrusions typically involve illegal and harmful requests that are sent to an application forcing it to execute out of its intended or authorized scope of operation. This may

exploit the application to damage itself, files, buffers, other applications, performance, or confidentiality of information.

Two conventional approaches attempt to address some of these problems. One technique involves tracking a server operating system to identify suspicious events such as deleting a file or formatting a disk. However, this type of reactionary technique typically activates only after damage has commenced or been completed. A second technique involves the installation of a network filter in front of an application and updating the filter database with known patterns that can affect the application. However, this technique is limited in that it is unable to identify patterns, which are not yet "known" by the filter database. In other words, the capability of this technique is directly related to the comprehensiveness of the filter database that it draws the patterns from. To increase capability, the filter database requires continual updating. Further, these techniques will not protect against manipulations of environmental variables or the application's implemented business process.

In addition, conventional security solutions typically fail to address the increased hacking opportunities caused by the proliferation of electronic commerce ("e-commerce"), mobile, and interactive television ("iTV") applications. These applications generally require the combination of numerous components operating on different platforms all working together using different technologies. For example, a single application can comprise a plurality of components, such as, a web server application; transaction server application; database; and Java, ActiveX, and Flash applets all working together. Generally, conventional security solutions are unable to meet the unique security needs of each component in a multiple component system.

SUMMARY OF THE INVENTION

The present invention overcomes these and other deficiencies of the related art by providing an application layer security method and system for preventing trusted computer applications and related resources from being accessed or executed by unauthorized users directly through the application itself.

In an embodiment of the invention, a method for protecting an application from executing an illegal or harmful operation request received from a

distrusted environment comprises the steps of applying one or more pipes to an incoming operation request. Illegal and harmful operation requests may cause damage or allow unauthorized access to the application, other applications, a trusted network, one or more data files, memory, buffers, performance, confidential information, hardware, software, a database, an information server, and the like. For example, illegal and harmful operation request may be database manipulations; attacks on known Internet information server vulnerabilities or application vulnerabilities; URL manipulations; business process manipulation; and the like. When an operation request is identified to be illegal or harmful, those request may be rejected entirely, or modified into or replace with a legal or harmless operation request.

In an embodiment of the invention, a security system routs application operation requests or results received or identified in a packet or stream of packets coming from the application server or from the application requester (client), before or after execution. In operation, an incoming packet or stream of packets is forwarded to an identifier that identifies one or more application commands or parameters, or a portion thereof, in an operation request. Identified commands or parameters are then forwarded to a router that comprises a routing scheme that directs each command or the entire request to an appropriate tunnel depending on, for example, the type of command, the application being requested, client identification, or a combination thereof. A feature of the invention is that different pipes can be applied to different parts of the operation request.

In an embodiment of the invention, a database driven pipe involves database operations and is aimed at finding patterns that show malicious intentions in a database request. Particularly, the contents of incoming database requests are inspected for permitted syntax based on the type of database used and the existence of one or more embedded harmful SQL commands.

In another embodiment of the invention, a deterministic type pipe inspects incoming operation requests for known vulnerability patterns. All known application vulnerabilities, and the respective requests that take advantage thereof, are stored and compared to the incoming request. If the comparison results in a match, the incoming request is blocked and is not allowed to proceed to the application. Further,

this pipe can shield applications against unknown vulnerabilities by eliminating the use of executables, methods, and scripts, which if accessed or executed, could negatively affect or allow illegal access to the application environment or servers.

In another embodiment of the invention, a HTML driven pipe ensures that an application operates as designed and implemented by the application owner. Particularly, the pipe checks each outgoing reply for its parameters names and values, then validates the next client based on a comparison between the incoming and outgoing parameters. This method ensures that subsequent user requests to the application are only those that the application expects.

In another embodiment of the invention, an application-node blocking pipe restricts remote user requests from entering application paths that are designated as restricted by an application owner. Application owners may sometimes keep applications running for fear of losing business, even though a vulnerability is known within the application, until a production patch is available. Using this pipe, application owners can block a vulnerable application path from being accessed while keeping other application paths available to clients.

In another embodiment of the invention, a pipe blocks distrusted operation requests from being forwarded to a trusted environment or zone within a trusted network. By implementing this pipe, an administrator can restrict a remote user's specific messages to only those that are designated as allowed.

In another embodiment of the invention, a pipe validates an incoming parameter value according to pre-defined expression rules.

In another embodiment of the invention, a cookie-based pipe protects client side cookies from being modified or manipulated by the user. This pipe ensures that internal information stored in the cookie will not be available to the user but will transparently be clear to the application without applying any change to the application itself.

In another embodiment of the invention, a SOAP pipe protects web services from servicing harmful client requests or preventing requests for services that remote clients or a specific client is unauthorized to call. Particularly, this pipe ensures that web services operate as designed and implemented by the application owner. This

pipe ensures that all client requests match the web service's definitions and data structures as well as allow access to only those services that remote clients are authorized to request.

In another embodiment of the invention, a learning technique allows the parameters in a request to be "fine tuned" to better match the most common values entered by the clients, i.e., requestors, determined from a statistical model. A dynamic range of entered values is built for each parameter in a request that a client sends. Based on the statistical mode implemented, a range of probable, i.e., most often entered or expected values is computed and the values of operation request parameters may be adjusted based on the reply from the application.

An advantage of the invention is that only legal and harmless requests will pass to a trusted network and to all the trusted applications therein. Another advantage is that the invention protects an unlimited number of applications installed on a single or multiple application servers without modifying or installing any software on these servers. Another advantage of the invention is that portions of an operation request can be scrutinized by different security techniques. Another advantage of the invention is that any combination of pipes can be applied to an application path according to the exact security needs of the application.

The invention provides a robust, scalable, dynamic, effective and elegant solution to the problem of attacks directed through the application layer.

The foregoing, and other features and advantages of the invention, will be apparent from the following, more particular description of the preferred embodiments of the invention, the accompanying drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, the objects and advantages thereof, reference is now made to the following descriptions taken in connection with the accompanying drawings in which:

Fig. 1 depicts the OSI model and the protocol layers that three conventional security techniques operate in;

Figs. 2A-2E illustrate an application layer security system according to an embodiment of the invention;

Fig. 3A-B illustrate an application layer security method according to an embodiment of the invention;

Fig. 4 illustrates an application layer routing system according to an embodiment of the invention;

Fig. 5 illustrates an application layer security system according to an embodiment of the invention;

Fig. 6 illustrates a first pipe according to an embodiment of the invention;

Fig. 7 illustrates a second pipe according to an embodiment of the invention;

Fig. 8 and **Fig. 9** illustrate a third pipe according to an embodiment of the invention;

Fig. 10 illustrates a fourth pipe according to an embodiment of the invention;

Fig. 11 illustrates a fifth pipe according to an embodiment of the invention;

Fig. 12 illustrates a sixth pipe according to an embodiment of the invention;

Fig. 13 and 14 illustrate a seventh pipe according to an embodiment of the invention; and

Fig. 15 illustrates an eighth pipe according to an embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Preferred embodiments of the present invention and their advantages may be understood by referring to **Figs. 2-15**. These embodiments may be employed in Internet, an intranet, a mobile communication network, an iTV network, a hybrid network, or any other application environment that uses application protocols, such as, but not limited to, HTTP, HTTPs, HTTPd, simple object access protocol ("SOAP"),

web distributed authoring and versioning ("WebDAV"), and simple mail transfer protocol ("SMTP"). Nevertheless, the inventive concept can be practiced in any type of communication system where information is exchanged between a trusted application or network and a distrusted communications environment. The inventive concept is particularly suited for protecting application service provider ("ASP"), business to commerce, and business to business network environments.

The invention prevents unauthorized use of a trusted computer application or resource requested directly through the application layer. The application layer is concerned with the requirements of the application and facilitates application operations requested by a client, as opposed to lower layers, which control the transmission of data between the client and the server. Generally, all application operations use elements, e.g., parameters and associated values, provided through the application layer. An application layer security method and system according to an embodiment of the invention implements one or more security connection techniques, referred to herein as pipes, either operating alone or in combination, comprising one or more application-layer security processes that scrutinize the application content of a client message, i.e., distrusted operation request, or incoming data stream to identify illegal or harmful commands to a trusted application or surrounding environment. Thus, protecting the trusted application against application layer, particularly, web related, threats. The application layer security method and system can be implemented to protect an application, a shared application, a distributed application, or multiple applications from being requested to execute out of the intended scope of operation. Thereby, preventing such applications from harming themselves, data files, buffers, other applications, server performance, or confidentiality of information. Operation requests that are identified to be harmful or illegal can be rejected entirely or modified into, or replaced with, a harmless or legal request. Accordingly, only harmless and legal requests will pass to the application for execution.

In an embodiment of the invention, a security system is positioned in front of a trusted environment comprising one or more applications. The security system implements a protective layer at the application layer between these trusted application(s) and incoming distrusted application operation requests that are received

from or through an unknown or distrusted environment. The protective layer inspects these requests for form, content, or both, to ensure that only legal and harmless requests will pass to the trusted environment. The security system may comprise hardware, software, or both, to implement the protective layer.

Fig. 2A illustrates an application layer security system 200 according to an embodiment of the invention. Particularly, protective layer 210 is implemented between a distrusted environment and a trusted environment comprising one or more applications (not shown) residing on one or more web servers 220. Protective layer 210 protects trusted applications from malicious operation requests sent via a distrusted source. Each incoming operation request can comprise a plurality of requested commands, parameters, and parameter values to be performed by an application. Incoming operation requests may be received as external HTTP traffic via one or more ports 230. Each application can be implemented on a dedicated or shared server, or distributed throughout multiple servers and can comprise resources, such as, but not limited to HTML documents, web server software, common gateway interface ("CGI") scripts, Perl scripts, database information, or any type of information resource or executable program. An application path represents an application's resource location to execute a specific application task or command. The resource location can be a physical or virtual address residing on web server 220. For example, **Fig. 2B** depicts the virtual subdirectory "/My App" consisting of subdirectory "/My App/Images" and "/My App/DOCS." Each one of these subdirectories represents an application path designating the location of respective images or documents.

Security system 200 comprises tunnels 240 to support single, distributed, or shared applications. Tunnels 240 are protocol based, e.g., TCP/IP, connections between the distrusted and trusted environments. Network traffic comprising incoming or outgoing data packets traveling through each tunnel is separated based on appropriate application paths derived from the application content of the data packets. For example, in HTTP, an application path can be a virtual directory defined on web server 220 that can be accessed via a specific tunnel. Each tunnel 245 can implement and validate a specified application protocol, such as, for example, HTTP, HTTPs, SOAP, WebDAV,

FTP, Telnet, and the like. Alternatively, each tunnel 240 can simply forward incoming and outgoing data packet streams.

Fig. 2C depicts exemplary tunnel 240 that communicates information between a listen address and port on the distrusted network side, and a connect local address on the trusted network side. The listen address and port receives incoming application operation requests. The connect local address receives information received through tunnel 240 and forwards it to the back-end application listening at its connect address and port. **Fig. 2D** depicts an exemplary tunnel implementation of tunnel 240, specifically referred to as tunnel #3, which is associated with subdirectory /DOCS. In this example, a request for the display of an application comprising certain documents, e.g., application command GET /DOCS in HTTP protocol, is transmitted through tunnel #3 to a corresponding application.

System 200 further comprises one or more pipes 250 provided to protect or monitor functionality of the applications. Particularly, one or more pipes 250 are executed on traffic running in a tunnel. For example, a first pipe protects an application(s) from known vulnerabilities, a second pipe prevents database manipulations, and a third pipe prevents parameters poisoning. Specific pipe embodiments and implementations are described in detail in the following description. One of skill in the art recognizes that additional pipes can represent other specific or general security concerns.

Referring to **Fig. 2A** and particularly **Fig 2E**, system 200 comprises one or more web applications 260, which are logical entities wrapping or grouping two or more tunnels 240, application paths, and one or more pipes 250 that together protect a trusted application. When protecting an application distributed among several web servers 220, i.e., separate HTML pages reside on different machines, a single web application entity represents the distribution. For example, web applications 260 scrutinize and monitor all traffic over selected application paths. A default web application is a logical entity representing security protection on all traffic over all tunnels not defined in web applications 260. Further, web applications 260 can customize pipe(s) execution to specific application paths.

Security system 200 can comprise hardware, software, or both to implement protective layer 210. In an embodiment of the invention, protective layer 210 is implemented on a stand-alone server. In an alternative embodiment, protective layer 210 is implemented as "plug and play" hardware, or the like, that can be installed on an existing server. In another embodiment of the invention, protective layer 210 is implemented by software components that are installed on one or more servers.

In an embodiment of the invention, an application layer security method comprises the step of inspecting the contents of incoming and/or outgoing messages or data packets to identify application commands or parameters associated with a trusted application executing out of its intended or authorized scope. Message contents comprise a message header, message body, query string, and the uniform resource identifier ("URI") and particularly the application layer elements, such as application variables and commands, which the application will execute or interpret. Content checking comprises the step of applying one or more pipes, wherein each pipe provides a specific protection or monitoring functionality, to the application layer content.

In an embodiment of the invention, a security method 300 is employed to scrutinize the application layer contents of incoming application operation requests. Referring to the left side of **Fig. 3A**, application path identification and tunnel construction is first performed (steps 302-322) on an incoming operation request. Particularly, an operation request in the form of a data packet or stream of packets sent from a distrusted network is first received (step 302) at a receiving means, such as, preferably a queued socket server, which prevents denial of services caused by buffer over-flow attacks, i.e., a large number of redundant or illegal requests that are received in a given time exceed the number that a server can handle, thereby causing a denial of service to legal requests or allowing overflow requests to proceed unauthorized. In other embodiments, other conventional receiving means able to receive and handle data communication may be used, the identification and implementation of which is apparent to one of skill in the art. Upon reception of the request, a binary stream reader reads (step 304) the data packets from the server queue. In a preferred embodiment, an internal entity, i.e., a network session object, handles each message during its lifetime in the system.

A protocol message constructor constructs (step 306) the received operation request, which may be initially formatted according to any type of protocol, into an internal message having a desired format designated by a protocol specification 307. Protocol specification 307 may be any type of application protocol, such as HTTP, HTTPs, SOAP, HTTPd, FTP, WebDAV, and the like, which is implemented by a trusted application. Internal message construction is also referred to as tunnel construction, where the operation request is embedded into a data format of the protocol of the trusted application. In an embodiment of the invention, if the operation request can not be formatted into an internal message because of, such as, the inability to create a legal protocol message due to, for example, missing or extra parameters, the operation request is immediately rejected. Upon successful formatting into an internal message, the message is indexed (step 308) and stored for later analysis, thereby permitting application content to be gathered from the request if necessary and accumulated in a temporary buffer or permanent storage device. As the following discusses in greater detail, gathered content is useful for comparison with a corresponding reply returned from an application's execution of the request or to subsequent other operation requests and replies.

If rejection does not occur, the message is translated (step 310) into a known encoding language that is understood by the pipe components. The known encoding language can be any type of encoding language. In a preferred embodiment of the invention, the encoding language used is Unicode or enhanced Unicode. Unicode is particularly preferable when using a HTTP application protocol. In operation, a request is translated into Unicode characters. For example, if a received request is encoded in American Standard Code for Information Interchange ("ASCII") or any other encoding standard, the message is translated into Unicode or enhanced Unicode. Translation provides an isolated and independent inspection of the operation request by determining if any symbols contained in the request are unexpected. In other words, any request that contains unexpected characters may be rejected in its entirety. Translation ensures that the subsequent security process is performed on an expected encoding standard, thereby minimizing the risk of security compromises due to unexpected characters. Moreover, translation isolates the original message from the translated message to ensure that all

pipes will execute properly on the translated message. In an embodiment of the invention, any update of protocol specification 307 leads to an immediate update of the known encoding language. In other embodiments, ASCII or any other encoding standard may be used as the encoding language.

Successful constructing, indexing, and translating results in the conversion of a distrusted operation request into a well-formatted protocol message encoded in Unicode. These steps act as a preliminary screening of the format of the request before the requests are passed along for intensive scrutiny of the contents by applying one or more pipes. Alternatively stated, these steps filter out requests that are blatantly surprising or unexpected before checking a request's contents for concealed or not readably identifiable illegal or harmful commands.

Following translation, each command or parameter, or a portion thereof, within an operation request is identified (step 312) along with its intended application path, and then related (step 314) to a logical managed entity, i.e., web application, employing a specified routing scheme. In an embodiment of the invention, a command is routed (step 316) according to the routing scheme if its identified application path is known, i.e., a match was found with an application path associated with the web application and routing scheme. If the command's application path does not exist or can not be completely identified, or does not match any known application path, the message is routed (step 318) through a default tunnel, which preferably applies all available pipes for maximum security. The default tunnel may also be applied to any message that was found to have a security problem during initial tunnel construction. Implementation of the default tunnel may trigger display of an error web page, security alert web page, or similar means to notify that an error has occurred or that a presumed malicious attack has been redirected away from the trusted application. However, if a default tunnel is not specified, then the request is denied (step 320). Each command or parameter corresponding to an appropriate application path is routed according to the routing scheme, thereby routing each command through a designated tunnel, which in turn applies a specified number and combination of pipes for that application path. Steps 312-320 are repeated for additional commands and parameters contained within the operation request (step 322).

Referring to the right side of **Fig. 3A**, one or more pipes are then applied sequentially (steps 324, 326, and 328), as determined by the routing scheme, to each incoming operation request to intensively scrutinize the application layer contents, such as, instructions, commands, query string, environmental variables, and parameters contained within the request. Each pipe provides a unique type of analysis or monitoring functionality. The specific number and types of pipes applied are dependent on the amount of security desired and typically for the type and needs of each tunnel and/or application path. For example, one tunnel may have three pipes applied, while another tunnel may have two applied.

Pipes are generally categorized as static or dynamic, either of which can optionally implement a "learning" technique as will be described. Static type pipes typically scrutinize the application contents through comparison with statically stored information that is related to hacking patterns or known, i.e., identified, vulnerabilities. For example, if a match is found through the comparison, the request can be rejected, modified, or replaced. Dynamic type pipes scrutinize both the incoming requests and outgoing replies, and the possibly changing, i.e., dynamic, relationship between them. Either of these type of pipes may apply a learning technique that gathers information from requests and/or replies to "fine tune" a pipe's scope of security. Nevertheless, a single pipe may combine a number of characteristics associated with static and dynamic type pipes, with or without a learning technique. Specific pipe configurations and implementations are described in the following description. Upon application of the appropriate number and types of pipes, the legality and/or harmfulness of the application contents of the operation request is determined. If the operation request passes scrutiny, the request is sent (step 330) or at least allowed to proceed to the trusted application for processing (step 332).

In a typical client-server communication session, once the application processes an operation request, a respective reply is generated for return to the requesting client. For example, the reply may be a HTML document, image, database information, video, or any other resource accessible through a server. Referring to **Fig. 3B**, the steps of method 300 that occur after the execution of the operation request are shown. The outgoing reply is read (step 334) and then constructed (step 336) into an

internal message based on a specified application protocol 337. In a preferred embodiment of the invention, application protocol 337 is identical to application protocol 307. If any of the pipes applied to the incoming operation request require inspection of the return reply, then steps 338-356 are performed. Particularly, the reply message is indexed (step 338) back to the original request if comparison between the two is necessary for any of the pipes implemented. Upon indexing the message, the reply message is translated to (step 340) the encoding language used on the incoming data stream. An outgoing reply is validated (step 342) using the translated indexed request stored previously and the translated and indexed reply message. Particularly, this step ensures that the outgoing reply is trivial in nature. For example, the reply may contain only information that is authorized for release from the server. A non-trivial reply is denied (step 344) from reaching the client. Allowable parameter names and values as well as new application paths are identified (respective steps 346 and 348) in the reply. This information is used to update appropriate dynamic type pipes or implemented learning techniques (steps 350, 352, and 354). Upon updating, the reply is reconstructed (step 356) into a message according to protocol 35, if necessary, and is sent (step 358) to the client requesting the operation. In a preferred embodiment, protocol 357 is identical to protocol 307. If the operation request did not have any pipes applied that require inspection of the reply, steps 338-356 are skipped.

Fig. 4 illustrates a security system 400 for routing application operation requests or replies received or identified in a packet or stream of packets coming from the application server or from the client. Particularly, system 400 comprises a client 410, a request identifier 420, a request router 424, N number of pipes 430A-N, a server 440, a reply identifier 450, and a reply router 454. Client 410 is presumed to be located in a distrusted network. Server 440 resides in a trusted network and hosts the interactions of client 410 with one or more applications (not shown) residing on server 440 or additional distributed servers (not shown). In operation, an incoming request comprising a packet or stream of packets transmitted by client 410 is forwarded to request identifier 420 via communications link 412. Request identifier 420 identifies one or more application commands or parameters, or portions thereof, in the request.

Identified commands or parameters are then forwarded to request router 424 via communications link 422.

Request router 424 directs each command or parameter, or the entire request itself, to an appropriate one of application paths 426A-M, where M is the number of unique or specified combinations of pipes 430A-N that may be applied. For example, as shown, application path 426A applies all N pipes to an operation request, whereas application path 426B applies only two pipes. However, any combination of pipes can be applied to each application path. The total number of application paths, M, and pipes, N, are not necessarily equal. In a preferred embodiment of the invention, request router 424 employs a routing scheme that determines the appropriate application path for each command or parameter based on, for example, the type of operation requested, the type of command, parameter name, parameter value, the specific application being requested, requested application path, client identification, or a combination thereof. Moreover, request router 424 may direct commands or parameters received from a specified or unidentifiable client address to a default application path that applies all available pipes. For certain pipes, such as pipe 430A, information ascertained from application of the pipe on a request, command, or parameter is used to dynamically modify the routing scheme employed by request router 424 via communications link 431. This routing scheme may be similarly modified by information ascertained by a pipe operating on an outgoing reply via communications link 463. Moreover, updates or modifications to the routing scheme may be received from a source (not shown) external to system 400. Upon application of all appropriate pipes to the sub-components, e.g., parameters and the like, of an operation request, the operation request is forwarded to server 440 for processing if all sub-components are deemed acceptable.

An outgoing reply comprising a packet or stream of packets, resulting from the processing of a request, undergoes similar scrutiny as the incoming stream. Particularly, the outgoing packet or stream of packets generated by server 440 is forwarded via communications link 442 to reply identifier 450, which identifies particular parameter names and values, or any other relevant application layer elements, contained therein. These identified parameters and elements are forwarded to reply

router 454 via communications link 452. Reply router 454 directs each element to an appropriate one of application paths 456A-M, where M is the number of unique or specified combinations of pipes 460A-N that may be applied to each outgoing parameter or element. A portion of pipes 460A-N can be identical to a portion of pipes 430A-N. In a preferred embodiment of the invention, reply router 454 employs a routing scheme that determines the appropriate application path for each request, parameter, parameter value, or element based on, for example, the type of reply, the type of parameter, the value of the parameter, the type of element, the application that generated the reply, identification of the client receiving reply, or a combination thereof. The reply routing scheme may be, but not necessarily, identical to the routing scheme employed in request router 424. Information ascertained by a pipe, such as pipe 460A as shown, on an outgoing parameter or element can be used to modify the routing scheme implemented by reply router 454 via a communications path 461. Moreover, information ascertained by pipe 430A operating on an incoming request can dynamically modify this routing scheme via communications path 433. Moreover, updates or modifications to the reply routing scheme may be received from a source (not shown) external to system 400. Upon application of all appropriate pipes 460 associated with application path 456, acceptable results are returned to client 410 via communications channel 462.

The above embodiment is illustrated with communication links 422, 433, 452, and 463 appropriately connected to request identifier 420, request router 424, reply identifier 450, or reply router 454. In an embodiment of the invention, identifier 420, request router 424, reply identifier 450, and reply router 454 may be implemented as software separate from pipes 430 and 460, which are implemented as either software or hardware. Therefore, communication links 422, 433, 452, and 463 are not actual physical connections, but virtual connections between software modules. Communications links 412, 432, 442, and 462, as well as any other link identified herein, may be any conventional communications medium, the implementation of which is apparent to one of skill in the art. In an alternative embodiment, request identifier 420, request router 424, reply identifier 450, and reply router 454 are implemented into a single hardware component with or without pipes 430A-N and 460A-N.

Fig. 5 illustrates a security system 500 having a hardware configuration according to an embodiment of the invention. Particularly, security system 500 comprises a security module 510, which is positioned in between trusted computer application 520 and a distrusted network environment. Module 510 implements a protective layer to prevent unauthorized use of application 520. In an embodiment of the invention, module 510 comprises a processor 530 and pipe logic 540. The term processor denotes any logic, circuitry, code, software, and the like that is configured to perform the functions described herein. Processor 530 performs initial tunnel construction, indexing, and translation as described above by implementing internal entities that isolate the operation requests received via the distrusted environment from application 520. The internal entities escort the distrusted requests to identifier and router 534 and analyzer 538. Identifier and router 534 applies every existing application path on the incoming request stream, or outgoing reply stream if necessary, to identify the operation request's desired destination in the application, as well as the parameters and values, if any. Analyzer 538 applies one or more pipes stored in pipe logic 540 to an operation request, or portion thereof.

EXEMPLARY PIPE DESCRIPTIONS

The following represents a detailed description of exemplary pipes that can be applied alone, or in combination, to incoming operation requests and outgoing replies. Additional pipes not mentioned herein, such as security processes directed toward conventional techniques, can be applied, the identity and implementation of which is apparent to one of skill in the art.

1. Marabu

In an embodiment of the invention, a first pipe, herein referred to as "Marabu," is applied to one or more application paths. Marabu is a content driven pipe aimed at finding patterns that show malicious intentions in an operation request. Marabu is preferably applied to application paths involving database operations. Particularly, Marabu inspects an incoming request's application contents for permitted syntax based on the type of requested database and the existence of one or more embedded harmful SQL commands. Harmful commands are identified by the failure to find a match to a plurality of stored database commands deemed acceptable for the type

of requested database, e.g., SQL type database or Oracle type database. If a match to an acceptable command is not found, the request will be rejected or reengineered, i.e., modified, into or replaced with an acceptable form that is legal or harmless. This pipe catches well-formed and partial SQL commands that are deemed to be harmful to the database or its environment.

In an embodiment of the invention, a database query parser engine is implemented to identify the form and function of each expression of the operation request. The parser engine parses out individual expressions, i.e., elements, for separate analysis. Marabu focuses on ensuring that the syntax of each expression is straightforward and credulous by applying a 'state-automate' and limited alphabet. Particularly, Marabu employs a method to enforce proper syntax comprising the steps of: building a state-automate; inspecting each and every expression according to permitted syntax and a limited alphabet, and applying the state-automate to each and every expression. The state-automate is a correlation of nodes representing present and other possible states of the application. Each node may have connections to other nodes based on the expressions that would change the state of the application. For example, a first node representing a present state of the application is connected to another node representing a possible state reached from the present state when a particular expression, e.g., a DELETE command, is executed. All connections are presumed to be one-way connections, i.e., once a state changes to another state, that change is irreversible.

In an embodiment of the invention, the alphabet contains: letters, digits, and other encoded characters, for example: a, b, c, _, ', etc.; blocks of letters, such as DELETE or ALTER; and groups of blocks, such as an SQL group comprising DELETE, ALTER and CREATE as members. Each expression within an operation request is scanned one or more times. If each expression consists of sequences of letters and digits identified as acceptable, the request is approved and pipe execution stops. If an expression does not consist of acceptable sequences of letters and digits, the remarks, particularly, the database specific character that represents a remark, such as MSSQL - /* */ or --, inside the expression is omitted and the expression is inspected by applying the state automate. Every predefined alphabet letter associated with an expression or a

function argument moves the current state of the automate to a connecting possible state.

In an embodiment of the invention, different processes are implemented depending on the type of state reached when applying the state automate. For example, if the state automate reaches a Boolean-state, the expression can be checked for "always-true" expressions, such as '15' = '1' + '5' and '6>5 OR 5=5'. Always-true expressions may trick a database into accepting a request when it normally shouldn't. Expression trees enable the identification of an always-true expression. Particularly, the expression is recursively broken into a tree having nodes as operators, e.g., AND, OR, =, and the like, or variables such as a field. After the tree is built, different expression values are inserted. A post-order scan will eventually lead to a TRUE or FALSE determination. Wisely sampling values to the variables will cover the range of all possible solutions, thereby determining if the expression is always true, and hence the rejection of the request. If the state automate reached a SQL-state, the expression is validated using SQL prototypes stored in a dictionary. The prototypes are implemented using a graph of different states connected by arcs. Two-way-arcs will allow recursive expressions, i.e., states with arcs pointing to themselves, and 'stretching' in order to try to fit the input. If the state automate reaches a trap-state, the expression is denied and execution of the pipe is halted. If none of these states occur or inspection of the above states does not result in denial of the expression, the expression is approved and pipe execution stops.

Fig. 6 illustrates a process 600 for implementing the Marabu pipe according to an embodiment of the invention. Particularly, configuration parameters for the supported database, e.g., based on the type of SQL version, are loaded (step 610) into a processor. Optionally, any refinements to these configuration parameters are loaded (step 620). The operation request is loaded (step 630) and parsed into elements to identify (step 640) expressions, e.g., parameter names and values, contained within the operation request. Marabu pipe is then applied (step 650) to each parameter to determine (step 660) if embedded SQL commands are acceptable. If not acceptable, the entire operation request can be rejected (step 670). If acceptable, it is determined (step 680) whether or not that parameter is the last parameter. If that parameter is not the last

parameter, steps 640-680 are repeated. If the parameter is the last parameter, and no illegal or harmful SQL commands are identified, the operation request is accepted (step 690) and allowed to proceed.

The following are two common types of database attacks that are generally wide spread across the Internet and are typical of manipulation type attacks, which can be caught by this pipe. The first represents a request to delete all records in a table. For example, in the delete all records attack, an original link, such as

http://www.yoursite.com/phones/phonelist.cgi?phoneid=34

which executes a SQL command on a database server to select and report the contents of "http://www.yoursite.com/phones" where "phoneid" equals 34, may be changed into a modified link concatenating a DELETE command, such as

http://www.yoursite.com/phones/phonelist.cgi?phoneid=34;DELETE

The latter executes a SQL command on the database to delete the contents of "http://www.yoursite.com/phones" where "phoneid" equals 34. In applying this pipe, the DELETE command is recognized as an unauthorized command and hence, may either reject the request entirely or replace the request with a legal request such as the original link not containing the DELETE command. In the following second type of illegal request, login is attempted without using a password. For example, an original link, such as

http://www.yoursite.com/logon.asp?login=yourname;pws=123

executes a SQL command on the database server to select a name from a user list where login='yourname' and the password='123'. A modified link adds an 'always true' Boolean phrase, to trick the application into accepting an invalid password, for example

http://www.yoursite.com/logon.asp?login=yourname' or '1' = '1' ; pws=8

executed a SQL command on the database server to select a name from a user list where login = 'yourname' or '1' = '1' and the password = '123'. When applying this pipe, the '1' = '1' partial command is recognized as invalid and the request is not allowed to achieve its intended execution.

Full database protection is provided against users and client applications manipulating an application database. Marabu pipe acts as a shield against data manipulation that can damage information stored within the database. Further, it

provides database system administrators with an additional level of protection when combined with other pipes and it ensures that application bugs will not affect a back office database.

2. Minime

In an embodiment of the invention, a second pipe, herein referred to as "Minime," is applied to one or more application paths. Minime is a deterministic pipe that inspects an incoming operation request for known vulnerability patterns. In a particular embodiment of the invention, all known application vulnerabilities, specifically the respective requests that take advantage thereof, are stored and compared to the incoming request. If a match if found between any portion of the incoming request and any portion of the information stored, the incoming request is blocked and is not allowed to proceed to the application. Further, the pipe may shield applications against unknown or not readily identifiable vulnerabilities by eliminating the use of scripts, commands, methods, and executables, e.g., format.exe, which formats a storage space, or cmd.exe, which enables a command-line input prompt, if accessed or executed, will negatively affect or allow illegal access to the application environment or servers.

In a preferred embodiment of the invention, the stored vulnerability information may be updated as future vulnerabilities are identified. Particularly, the stored information may be updated with information downloaded from an information source, for example, a dedicated centralized server or multiple web servers from commercial or government sources.

In an embodiment of the invention, the Minime pipe employs an exact pattern matching method. Particularly, the method comprises the steps of: computing a hash value for consecutive characters, e.g., for every four (4) consecutive characters, in an operation request and comparing each computed hash value against a list of hash values associated with improper requests. In operation, the first four characters of an operation request is inputted into a four (4) character window. Each of the four characters is converted to an 8 bit binary code, for example:

abcd = [01000001][01000010][01000011][01000100]

A first hash value is then calculated from the 32 bit string (4 characters x 8 bits/character). A second hash value is calculated by shifting the window one (1) character, i.e., characters 2 through 5. A third hash value is calculated by shifting the window again by one character, i.e., using characters 3 through 6. Additional hash values are calculated in such a way until all the characters in the operation request have been exhausted. Thus, if there are n characters in an operation request, n-3 has values will be calculated. Each hash value is compared to those stored and associated with improper requests. Accordingly, if a match is found, the incoming request is rejected.

In a preferred embodiment of the invention, the above method is applied to an operation request constructed in HTTP protocol. The operation request is divided into four (4) zones respectively representing a message URI, a query string, a message header, and a message body. Vulnerability pattern matching as described above is directed to one or more of the specific four zones. Because pipe application can be directed to a single zone or a portion less than the entire operation request itself, performance can be increased and false alarms decreased. If a string is determined to be a suspect, full simple string matching is performed.

Fig. 7 illustrates a process 700 for implementing the Minime pipe according to an embodiment of the invention. Particularly, vulnerability patterns are loaded (step 710) into a processor either from internal storage or an external database. Optionally, a table comprising has values of the vulnerability patters is created (step 720) to facilitate efficient pattern matching. Four different scanning zones are built (step 730) and associated with designated hash values to be applied. The incoming message is loaded (step 740) and one or more message zones are scanned (step 750) to determine (step 760) if a vulnerability pattern, e.g., hash value, matches. If a match is found, the message can be rejected (step 770). If a match is not found, the message is allowed (step 780) to proceed to another pipe or an application. For any subsequent operation request, steps 740-780 are repeated.

The following are three common Microsoft Internet Information Server® vulnerabilities blocked. View any file on your web server hard disk, e.g.,

http://www.yoursite.com/msadc/Samples/SELECTOR/

showcode.asp?source=/msadc/Samples/../../../../../boot.ini.

Execute any SQL command directly to your database, e.g.,

http://www.yoursite.com/msadc/samples/adctest.asp.

View Outlook mail folders using CodeBrws.asp, e.g.,

http://www.yoursite.com/iissamples/exair/howitworks/codebrws.asp?source=/../../winnt/Profiles/Administrator/Application%20Data/Microsoft/Outlook%20Express/Mail/inbox.mbx

The Minime pipe provides at least the following benefits: application protection from remote users attempting to hack a system using application vulnerabilities; temporary protection to applications until a patch is available and installed on the server; and custom patterns can be added and existing patterns can be removed from storage.

3. Hide & Seek

In an embodiment of the invention, a third pipe, herein referred to as "Hide & Seek," is applied to one or more application paths. Hide & Seek is a hypertext markup language driven pipe whose purpose is to ensure that an application operates as designed and implemented by the application owner. Hypertext markup language or HTML herein includes any type of markup language, such as, extensible markup language ("XML"), extensible hypertext markup language ("XHTML"), and wireless markup language ("WML"). Particularly, the pipe checks each outgoing HTML reply to identify its internal parameters names and values, and then validates the next user request corresponds to the previously identified parameters, thereby ensuring that subsequent user requests to the application are only those that the application expects or intended to be returned.

In an embodiment of the invention, a method comprises identifying a single client to interact with the application, identifying elements, e.g., parameter names and values, that were sent to the client in a reply from the application, determining whether a client's subsequent request to the application includes any parameter and values, and if so, determine whether those parameters and values are expected by the application. Only if the parameters and values received from the client correspond to the elements determined from the preceding, corresponding message sent will the server forward the request to the application.

In an embodiment of the invention, an algorithm performs HTML parsing on a reply document and stores all HTML tags that are received by the client. For example, a parser reads each reply coming from an application and identifies specific HTML elements, e.g., input, select, form, etc., in the reply message. For each identified element the parser determines the element value. All parsed element names and values are stored in a session object located on the server, wherein each session is related to a single client. After identifying the client and optionally attaching to the request a session object, an additional parser operates on the request. At this stage, a comparator is used to match the request with the reply based on the HTML elements. For example, the reply parser determines whether the client can change a parameter value on his next request. If so, it permits a client to send any value with this parameter. For example, if the application reply includes an <INPUT> tag, the client is expected to enter a value.

In order to identify a client in a state-less protocol, i.e., a protocol that is only concerned with the source and destination information, but not the contents of the message, a session-based cookie is utilized. Preferably, cookies are enabled on a client's web browser. For security purposes, a client without cookies enabled is considered as a 'new' or unauthorized user who is limited to minimal application activities. If cookies are enabled by the client, the server generates and stores a session-based cookie at the client upon initiation of a client-server session. In an embodiment of the invention, the session-based cookie comprises a header comprising client identification information and a cookie body comprising an encrypted security value identifying a session object.

The following is an exemplary implementation of the Hide & Seek pipe. A typical HTML document for login and password is as follows:

```
<HTML><BODY BGCOLOR="#FFFFFF">
<FORM name="faq" method=POST action="login.asp" >
<BR><BR><BR><BR><CENTER><TABLE><TR>
<TD>Login: </TD>
<TD>Password: </TD>
</TR><TR>
<TD><INPUT TYPE="text" NAME="Login"></TD>
<TD><INPUT TYPE="password" NAME="password"></TD>
<TD> <INPUT TYPE="hidden" NAME="state" value=0></TD>
```

```
</TR></TABLE><BR><BR>
<INPUT TYPE="submit" name="Submit">
</CENTER></FORM>
<FORM name="remember" method=POST action="remember.asp" >
<BR><BR><BR><BR><CENTER><TABLE><TR>
<TD>Login: </TD></TR><TR>
<TD><INPUT TYPE="text" NAME=" RememberPWS "></TD>
<TD> <INPUT TYPE="hidden" NAME="state" value=2></TD>
</TR></TABLE><BR><BR>
<INPUT TYPE="submit" name="Submit"></CENTER>
<A HREF="/Terms.asp?x=1&y=3">Terms</A>
</FORM></BODY></HTML>
```

This HTML document is sent to the client upon the client first accessing a server. Upon parsing, the following exemplary results are stored in a session object:

```
[Group]=1
Login=[text]
Password=[text]
State=0
Submit=[text, not a must]
[Group]=2
RememberPWS=[text]
State=2
Submit=[text, not a must]
[Group]=3
X=1
Y=3
```

This session object is associated with a session value, which is stored in a cookie at the client. On the client's next subsequent request, i.e., the client's response to the server asking for a login and password, Hide & Seek obtains the session value from the client and identifies the session object associated with the session value. The pipe then obtains the results stored within the session object and validates the incoming parameter names and values against those results. If the incoming and outgoing parameters match, then the client is presumed to have not manipulated any of the parameters. However, if any of the incoming and respective outgoing parameters do not match, the user is presumed to have manipulated the request. Therefore, the incoming request is rejected.

In an embodiment of the invention, a refinement declaration table is employed to handle text fields, e.g., login field. The table includes parameters and its expected data type, length, allow null, etc.

Fig. 8 illustrates a process 800 for implementing the Hide&Seek pipe according to an embodiment of the invention. Particularly, configuration parameters are loaded (step 805) according to a supported HTML version. Refinements are then optionally loaded (step 810) if desired. An incoming operation request message is loaded (step 815) for analysis and then checked (step 820) for client identification in a HTTP cookie header. Subsequently, the body of the cookie is obtained (step 825) and decrypted (step 830) to get a session value. A session object associated with the session value is obtained (step 835). If no session object exists, a blank or new one is created. Then, the incoming message is parsed (step 840) into parameters name and values. Parameters, values, and combinations thereof are compared (step 845) to stored information in the session object or to the refinement table. An existence of a match is then determined (step 850). If a match is not found, the message is rejected (step 855). If all parameters match the message is allowed (step 860) to proceed. Steps 815-860 are repeated on subsequent incoming requests.

Fig. 9 illustrates a process 900 for implementing the Hide&Seek pipe on a reply message according to an embodiment of the invention. First, a reply message is received (step 910) and loaded (step 920) to determine (step 930) if the reply is an HTML document. If not HTML, the reply message is sent (step 940) to the client. If HTML, the reply message is parsed (step 950) into parameters. All parsed information is stored (step 960) in a session object. The session object is identified (step 970) as whether or not it is a new basket. If session object is not new, the message is sent (step 980) to the client. If the session object is new, a header is added (step 990) to the reply message comprising information identifying the session and then the entire message is then sent (step 995) to the client.

Hide & Seek protects applications from remote users manipulating an application business process. Further, it protects the application from any requests involving a manipulation technique and provides application owners with immediate and continuous shielding.

4. Inigo

In an embodiment of the invention, a fourth pipe, herein referred to as "Inigo," is applied to one or more application paths. Inigo is an application-node

blocking pipe, which restricts application operation requests from entering application paths that are designated as restricted by an application owner. In an environment where business applications are running, application owners are typically faced with an administrative problem when a bug was found associated with an application path. However, stopping all execution of an application associated with that application path negatively affects one or more business processes the application implements. Therefore, application owners typically keep applications running, even though a vulnerability is known within the application, until a production patch is available. By implementing the Inigo pipe, application owners can block a vulnerable application path from being accessed while keeping other applications available to users.

Fig. 10 illustrates a process 1000 for implementing the Inigo pipe according to an embodiment of the invention. Application paths are loaded (step 1010) from a data source. An incoming request message is loaded (step 1020) into logic. The destination of the message is then determined (step 1030) and compared (step 1040) against specified application paths designated as blocked. If a match is found, the message is rejected (step 1060). If a match is not found, the message is allowed (step 1050) to pass.

For example, to block a press release directory from browsing, due to incorrect published information on the link:

http://www.yoursite.com/mysite/press/news.asp?newsid=23,

Indigo identifies requests for this application path and either rejects the request entirely or redirects the request so that a different press release, such as

http://www.yoursite.com/mysite/default.html,

is returned to the client.

The Inigo pipe provides application owners with immediate traffic blocking capabilities without waiting for production pipes to be implemented on all servers, thereby easing the complexity of administrating a single or large amount of applications when a vulnerability is found.

5. Ogini

In an embodiment of the invention, a fifth pipe, herein referred to as "Ogini," is applied to one or more application paths. Ogini is a pipe that blocks

distrusted operation requests from being forward to a trusted environment or zone within a trusted network. By implementing this pipe, an administrator can restrict a client's specific messages to only those application paths that are designated as allowed. Particularly, a pre-defined list of allowed messages is implemented. If a client's request message matches any in the list, the operation request will be accepted. Otherwise, the operation request is rejected. To provide administrators with an easy way to update the allowed message list, the pipe optionally supports a learning technique, where all incoming request messages determined to be acceptable are stored in a memory for later comparison to incoming requests.

Fig. 11 illustrates a process 1100 for implementing the Ogini pipe. Particularly, a list of acceptable operation requests is loaded (step 1110) into a processing means. The incoming message is loaded (step 1120) and compared (step 1130) to the list to determine (step 1140) if a match exists. If a match is found, the operation request is allowed (step 1150) to pass. Otherwise, the operation request is rejected (step 1160).

6. Miloba

In an embodiment of the invention, a sixth pipe, herein referred to as "Miloba," is applied to one or more application paths. Miloba is a pipe that validates an incoming parameter value according to pre-defined expression rules. For example, in an online bookstore, customers may be allowed to order one to five books in each order. The bookstore uses an order_qty parameter to represent the number of ordered books. Miloba verifies that the user order_qty parameter value will match an integer data type with a value between one and five. In an embodiment of the invention, in order to identify and construct the pre-defined expression rules for each parameter, learning techniques are implemented in a continuous process to identify new acceptable parameters within an operation request by updating the pre-defined rules based on previously submitted operation requests.

Fig. 12 illustrates a process 1200 for implementing the Miloba pipe according to an embodiment of the invention. Pre-defined rules are loaded (step 1210) from a data source. An incoming operation request is loaded (step 1220) into a processing means. Parameter names and values contained within the operation request

are determined (step 1230). For each type of parameter, a particular rule associated with that parameter is identified (step 1240) form the pre-defined rules and then applied (step 1250) to the incoming parameter value. Accordingly, it is determined (step 1260) whether the rule is satisfied. If satisfied, the parameter value is accepted (step 1270). Otherwise, the operation request is rejected (step 1280).

7. Cookie

In an embodiment of the invention, a seventh pipe, herein referred to as "Cookie," is applied to one or more application paths. This pipe protects client side (session based or persisted) cookies from being modified or manipulated by a client. The pipe ensures that information stored in the cookie is not available to the client, but will transparently be clear to the application without applying any change to the application itself. Particularly, cookie identification, which identifies a cookie that is associated with the request and its location, is encrypted on the client side to prevent a client from determining the identification of the cookie. Any type of encryption method can be employed, implementation of which is apparent to one of skill in the art.

Fig. 13 illustrates a process 1300 for implementing the Cookie pipe on an incoming operation request according to an embodiment of the invention. An incoming operation request is loaded (step 1310) into a processor. The pipe searches (step 1320) for cookie identification as defined by a protocol specification in the operation request. Then, the pipe determines (step 1330) whether any cookie identification was found. If found, the cookie information is decrypted and the operation request is modified (step 1340) by substituting the decrypted cookie information for the encrypted cookie information. The cookie is then forwarded (step 1350) to the next security pipe. If cookie information is not found, then step 1340 is not performed.

Fig. 14 illustrates a process 1400 for implementing the Cookie pipe on a reply message according to an embodiment of the invention. The reply is loaded (step 1410) and scanned (step 1420) for a cookie identification as defined by the protocol specification. If cookie identification was found (step 1430) that information is encrypted (step 1440). The reply is then subsequently modified to reflect the encrypted value. The message is then forwarded (step 1450) to the next security pipe or the client.

8. SOAP

In another embodiment of the invention, a SOAP pipe protects web services from servicing harmful client requests or preventing requests for services that remote clients or a specific client is unauthorized to call. Particularly, this pipe ensures that web services operate as designed and implemented by the application owner. This pipe ensures that all client requests match the web service's definitions and data structures as well as allow access to only those services that remote clients are authorized to request.

Fig. 15 illustrates a process 1500 for implementing the SOAP pipe on an incoming operation request according to an embodiment of the invention. Pre-defined rules are loaded (step 1510) from a data source. An incoming operation request is loaded (step 1520) into a processing means. Service name, parameter names, type and values contained within the operation request are determined (step 1530). For each service name, a web service definition is fetched (step 1540). Upon fetching, if the no service definition is found (step1550), the operation request is rejected (step 1590). If the service definition is found (step 1550), a particular rule associated with that service is identified and then applied (step 1560). Accordingly, it is determined (step 1570) whether the rule is satisfied. If satisfied, the operation request is accepted (step 1580). Otherwise, the operation request is rejected (step 1590).

9. Learning

In an embodiment of the invention, a learning technique is employed in one or more pipes applied on one or more application paths. The application of a learning technique allows the parameters in a request to be "fine tuned" to better match the most common values entered by the clients, i.e., requestors, determined from a statistical model. Particularly, all operation requests to each application path are gathered and stored into a virtual storage. A dynamic range of entered values is then built for each parameter in a request that a client sends. Based on the statistical mode implemented, a range of probable, i.e., most often entered or expected values is computed. Further, the values of the parameters may be adjusted based on the reply from the application.

For example, consider a database operation request that it is requesting that the contents of a stored field "F" be reported back to the requestor. A specified statistical model determined that ninety (90) percent of the requests ask for a particular F within the range of nine (9) to eighteen (18). Further, it is desired to limit F to within this range. In this example, any F value between nine (9) and eighteen (18) will be allowed to pass to the application and executed, however, a F value of twenty (20) will either be rejected totally, or adjusted to another value, such as the closest acceptable value, e.g., 18.

This technique is referred to as a learning technique because of it's ability to adjust or "learn" from the information accumulated from the parameter values and/or application paths requested over time. In other words, although the statistical model may not change, the range of the most common values dynamically changes with the values "learned" from numerous request operations. For example, the boundaries of the most common values may change as more and more values are received that are outside the original range of common values. The learning technique provides another method to strengthen the application layer protection. The learning operation can be applied on each web application separately or on all available tunnels.

In an embodiment of the invention, a web application comprises two modes of operation, e.g., a learn mode or an enabled mode. In the learn mode, the web application employs a method to collect all security errors determined by the pipes. For each error, the operation request's properties, such as, web application, tunnel, application path, and requested application; the operation message parameters; and the pipe that determined this security event are recorded. A refinement recommendation record for the relevant pipe is created at the relevant pipe. A refinement recommendation record comprises configuration information that the learning technique provides to the system administrator for application to each pipe. In the enabled mode of operation, for each incoming operation request, the web application determines if the operation request is an approved message by passing it to the pipe(s). If approved, the operation request is deemed to be trusted. If not, the operation request is passed along for additional pipe inspection.

In order to identify each parameter related regular expression, the learning technique uses a statistical normal distribution that accumulates each parameter value. Based on the current variance, the algorithm refines the current parameter regular expression validation process. For example, operation requests include a price parameter, such as:

Step	Parameter value	Regular expression
1	Price=12$	Long [0-1e6],[a-z],[./'@$]
2	Price=15	Int [0-100],[/@$]
3	Price=34$	Int[5-50],[$]
4	Price=25.5$	Long[5-50],[.$]
X	Price=22.2$	Long[10-30][.$]

The x step, represents the final regulation expression, the learning techniques has. A final stage is reached when the distribution variance of the parameter regulation expression is low.

The invention can complement conventional security technologies. For example, in an embodiment of the system, a firewall is included in between the distrusted network and the security system allowing the firewall to implement its security methods at the data link layer on distrusted operation requests before they are scrutinized at the application layer. In other embodiments, the inventive concept can be combined with any lower layer security technologies, such as, SSL or PKI.

Although the invention has been particularly shown and described with reference to several preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the spirit and scope of the invention as defined in the appended claims.

CLAIMS

I claim:

1. A method for protecting an application from executing an illegal or harmful operation request received from a distrusted environment, the method comprising the steps of:

applying one or more pipes to an application-layer contents of an operation request to determine if said operation request is illegal or harmful to an environment of said application, and

preventing said application from executing an illegal or harmful operation request.

2. The method of claim 1, wherein said illegal and harmful operation request causes damage or allows unauthorized access to a computer system element or parameter selected from the group consisting of: said application, an application other than said application, a trusted network, one or more data files, memory, buffers, performance, confidential information, hardware, software, a database, an information server, and any combination thereof.

3. The method of claim 1, wherein said illegal and harmful operation request is selected from the group consisting of:

a database manipulation, an attack on an Internet information server vulnerability, an attack on an application vulnerability, a URL manipulation, an application request manipulation, a business process manipulation, a poisoning attack, an application parameters manipulation, and any combination thereof.

4. The method of claim 1, wherein said step of preventing comprises the step of

rejecting said illegal or harmful operation request.

5. The method of claim 1, wherein said step of preventing comprises the step of

modifying said illegal or harmful operation request into a legal or harmless operation request.

6. The method of claim 1, wherein said step of preventing comprises the step of

replacing said illegal or harmful operation request with a legal or harmless operation request.

7. The method of claim 1, wherein one of said one or more pipes comprises the step of checking for an existence of an embedded command causing a database manipulation.
8. The method of claim 7, wherein said embedded command is a portion of a SQL based command.
9. The method of claim 7, wherein one of said one or more pipes comprises the steps of:

parsing said operation request into one or more expressions;

building a state-automate;

inspecting said one or more expressions for improper syntax and characters not defined in a first alphabet; and

applying said state-automate to said operation request.
10. The method of claim 9, wherein said alphabet is selected from the group consisting of: letters, digits, and encoded characters; blocks of letters; groups of said blocks; and any combination thereof.
11. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

comparing said operation request against stored known vulnerability patterns to determine a match; and

blocking said operation request if a match is found.
12. The method of claim 11, further comprising the step of:

updating said stored vulnerability patterns with additional vulnerability patterns.
13. The method of claim 11, wherein said step of comparing comprises the steps of:

converting every consecutive specified number of characters in said operation request into n-bits of binary code;

computing a hash value for said every consecutive specified number of characters in said operation request; and

comparing every hash value to stored hash values representing vulnerability patterns.

14. The method of claim 13, wherein n is eight (8) and said specified number is four (4).

15. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

dividing said operation request into zones;

comparing each of said zones against stored known vulnerability patterns to determine a match; and

blocking said operation request if a match is found.

16. The method of claim 15, wherein said zones comprise:

a first zone consisting of a URI of said operation request,

a second zone consisting of a query string of said operation request,

a third zone consisting of a header of said operation request, and

a fourth zone consisting of a body of said operation request.

17. The method of claim 1, further comprising the steps of:

sending a legal or harmless operation request to said application; and

receiving a reply to said legal or harmless operation request.

18. The method of claim 17, wherein one of said one or more pipes comprises the steps of:

determining parameters names and values contained in said reply;

receiving a response to said reply;

comparing said response to said parameters names and values to determine if said response is a valid response to said reply; and

disallowing said response to pass to said application if said response is not a valid response to said reply.

19. The method of claim 17, further comprising the steps of:

identifying a single client to interact with said application;

determining a first set of parameter names and values in said reply;

receiving a second operation request from said client in response to said reply; and

determining parameter names and values in said second operation request.

20. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

designating an application path of said application as restricted;

determining a destination of said operation request; and

blocking said operation request if said destination is equal to said designated application path.

21. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

compiling a list of acceptable operation requests; and

comparing said operation request to said list of acceptable operation requests.

22. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

determining a parameter value contained within said operation request; and

applying a pre-defined rule to said parameter based on said parameter type, wherein said pre-defined rule defines one or more acceptable parameter values.

23. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said operation request;

decrypting a portion of said cookie; and

modifying said cookie to reflect said decrypted portion.

24. The method of claim 17, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said reply;

encrypting a portion of said cookie; and

modifying said cookie to reflect said encrypted portion.

25. The method of claim 1, further comprising the steps of:

receiving a plurality of operation requests;

storing said plurality of operation requests into a virtual storage;

building a dynamic range of entered values for each parameter in said plurality of operation requests;

computing an acceptable range of values for each parameter based on a statistical model applied to said dynamic range of entered values for each value;

receiving a subsequent operation request;

identifying parameter values in said subsequent operation request; and

determining if said parameter values in said subsequent operation request are within said acceptable range of values.

26. The method of claim 25, further comprising the steps of:

adding said parameter values in subsequent operation requests to said dynamic range; and

adjusting said acceptable range of values for each parameter by applying said statistical model.

27. The method of claim 1, wherein one of said one or more pipes comprises the steps of:

identifying a service name of said operation request;

loading a web service definition associated with said service name; and

applying a rule based on said web service definition to said operation request.

28. A method for preventing an application from executing out of its authorized scope of operation, comprising the steps of:

identifying one or more parameters in an operation request;

resolving an application path for each parameter; and

applying one or more pipes to each parameter, wherein the number and types of pipes applied to each parameter is based on said resolved application path of parameter.

29. The method of claim 28, further comprising the steps of:

formatting said operation request into a formatted message according to a designated communications protocol;

indexing said formatted message;

storing said formatted message; and

translating said formatted message into an internal message according to an encoding scheme, wherein said designated communications protocol is selected from the group consisting of: HTTP, HTTPs, HTTPd, WebDAV, and SOAP.

30. The method of claim 29, wherein said encoding scheme is Unicode.

31. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

parsing an operation request into one or more expressions;

building a state-automate;

inspecting said one or more expressions for improper syntax and characters not defined in a first alphabet; and

applying said state-automate to said operation request.

32. The method of claim 31, wherein said alphabet is selected from the group consisting of: letters, digits, and encoded characters; blocks of letters; groups of said blocks; and any combination thereof.

33. The method of claim 28, wherein one of said one or more pipes comprises the steps of

comparing said operation request against stored known vulnerability patterns to determine a match; and

blocking said operation request if said match is found.

34. The method of claim 33, further comprising the step of:

updating said stored vulnerability patterns with additional vulnerability patterns.

35. The method of claim 28, wherein said step of comparing comprises the steps of:

converting every consecutive specified number of characters in said operation request into n-bits of binary code;

computing a hash value for said every consecutive specified number of characters in said operation request; and

comparing every hash value to stored hash values representing vulnerability patterns.

36. The method of claim 35, wherein n is eight (8) and said specified number is four (4).

37. The method of claim 28, wherein one of said one or more pipes comprises the steps of

dividing an operation request into zones;

comparing each of said zones against stored known vulnerability patterns to determine a match; and

blocking said operation request if said match is found.

38. The method of claim 37, wherein said zones comprise:

a first zone consisting of a URI of said operation request,

a second zone consisting of a query string of said operation request,

a third zone consisting of a header of said operation request, and

a fourth zone consisting of a body of said operation request.

39. The method of claim 28, further comprising the steps of:

sending a legal or harmless operation request to said application; and

receiving a reply to said legal or harmless operation request.

40. The method of claim 39, wherein one of said one or more pipes comprises the steps of:

determining a first set of internal application paths and parameters values contained in said reply;

receiving a response to said reply;

determining a second set of application paths and parameters values contained in said response;

comparing said second set with said first set to determine if said response is a valid response to said reply; and

disallowing said response to pass to said application if said response is not a valid response to said reply.

41. The method of claim 39, wherein one of said one or more pipes comprises the steps of:

identifying a single client to interact with said application;

determining a first set of parameter names and values in said reply;

receiving a second operation request from said client in response to said reply;

determining a second set of parameter names and values in said second operation request; and

forwarding said second request to said application only if said first set matches said second set.

42. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

designating an application path of said application as restricted;

determining a destination of said operation request; and

blocking said operation request if said destination is equal to said designated application path.

43. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

compiling a list of acceptable operation requests; and

comparing said operation request to said list of acceptable operation requests.

44. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

determining a parameter value contained within said operation request; and

applying a pre-defined rule to said parameter based on said parameter type, wherein said pre-defined rule defines one or more acceptable parameter values.

45. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said operation request;

decrypting a portion of said cookie; and

modifying said cookie to reflect said decrypted portion.

46. The method of claim 39, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said reply;

encrypting a portion of said cookie; and

modifying said cookie to reflect said encrypted portion.

47. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

storing said plurality of operation requests into a virtual storage;

building a dynamic range of entered values for each parameter in said plurality of operation requests;

computing an acceptable range of values for each parameter based on a statistical model applied to said dynamic range of entered values for each value;

receiving a subsequent operation request;

identifying parameter values in said subsequent operation request; and

determining if said parameter values in said subsequent operation request are within said acceptable range of values.

48. The method of claim 47, further comprising the steps of:

adding said parameter values in subsequent operation request to dynamic range; and

adjusting said acceptable range of values for each parameter by applying said statistical model.

49. The method of claim 28, wherein one of said one or more pipes comprises the steps of:

identifying a service name of said operation request;

loading a web service definition associated with said service name; and

applying a rule based on said web service definition to said operation request.

50. A system for implementing an application security layer between an application and a distrusted computer environment comprising:

means for receiving an application operation request, wherein said operation request comprises one or more parameters;

an identification module for identifying said one or more parameters; and

a router module for routing each of said one or more parameters to one or a plurality of application paths according to a routing scheme.

51. The system of claim 50 further comprising:

means for formatting said operation request into a data format used by said application.

52. The system of claim 51, wherein said data format is selected from the group consisting of:

HTTP, HTTPs, HTTPd, WebDAV, and SOAP.

53. The system of claim 50, wherein said receiving means is a queued socket server.

54. The system of claim 50, further comprising:

logic implementing one or more pipes, wherein a number of said one or more pipes is applied to each application path.

55. The system of claim 54, wherein said number depends on each application path.

56. A system for implementing an application security layer between an application and a distrusted computer environment comprising:

means for receiving an operation request for an application;

means for resolving a destination application node for said operation request; and

means for applying one or more pipes to said operation request, wherein the number and types of pipes applied to said operation request are based on said resolved destination node of said operation request.

57. The system of claim 56, further comprising

means for formatting said operation request into a data format used by said application, wherein said data format is selected from the group consisting of:

HTTP, HTTPs, HTTPd, WebDAV, and SOAP.

58. The system of claim 56, further comprising

an encoder for encoding said operation request according to an encoding scheme, wherein said encoding scheme is Unicode.

59. The system of claim 56, further comprising a firewall.

60. A computer-readable medium comprising computer-executable instructions thereon for preventing one or more applications from executing out of their intended scope of operation, the instructions facilitating the steps of:

identifying one or more parameters in an application operation request;

resolving an application path for each parameter; and

applying one or more pipes to each parameter, wherein the number and types of pipes applied to each parameter is based on said resolved application path of parameter..

61. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the step of

routing said parameter to said resolved application path.

62. The computer-readable medium of claim 60, the instructions further facilitating the step of

routing said parameter to a default application path if said application path is not resolved.

63. The computer-readable medium of claim 60, the instructions further facilitating the steps of:

formatting each operation request into a formatted message according to a designated communications protocol;

indexing said formatted messages;

storing copies of said formatted messages; and

translating said formatted messages into internal messages according to an encoding scheme, wherein said encoding scheme is Unicode.

64. The computer-readable medium of claim 63, wherein said designated communications protocol is selected from the group consisting of: HTTP, HTTPs, HTTPd, WebDAV, and SOAP.

65. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

parsing an operation request into one or more expressions;

building a state-automate;

inspecting said one or more expressions for improper syntax and characters not defined in a first alphabet; and

applying said state-automate to said operation request.

66. The computer-readable medium of claim 65, wherein said alphabet is selected from the group consisting of: letters, digits, and encoded characters; blocks of letters; groups of said blocks; and any combination thereof.

67. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

comparing said operation request against stored known vulnerability patterns to determine a match; and

blocking said operation request if said match is found.

68. The computer-readable medium of claim 60, the instructions further facilitating the step of

updating said stored vulnerability patterns with additional vulnerability patterns.

69. The computer-readable medium of claim 65, wherein said step of comparing comprises the steps of:

converting every consecutive specified number of characters in said operation request into n-bits of binary code;

computing a hash value for said every consecutive specified number of characters in said operation request; and

comparing every hash value to stored hash values representing vulnerability patterns.

70. The computer-readable medium of claim 69, wherein n is eight (8) and said specified number is four (4).

71. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

dividing an operation request into zones;

comparing each of said zones against stored known vulnerability patterns to

determine a match; and

blocking said operation request if said match is found.

72. The computer-readable medium of claim 71, wherein said zones comprise:

a first zone consisting of a URI of said operation request,

a second zone consisting of a query string of said operation request,

a third zone consisting of a header of said operation request, and

a fourth zone consisting of a body of said operation request.

73. The computer-readable medium of claim 60, the instructions further facilitating the steps of:

sending a legal or harmless operation request to said application; and

receiving a reply to said legal or harmless operation request.

74. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

determining a first set of internal application paths and parameters values contained in said reply;

receiving a response to said reply;

determining a second set of internal application paths and parameters values contained in said response;

comparing said second set with said first set to determine if said response is a valid response to said reply; and

disallowing said response to pass to said application if said response is not a valid response to said reply.

75. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

determining a first set of parameter names and values in said reply;

receiving a second operation request from said client in response to said reply;

determining a second set of parameter names and values in said second operation request; and

forwarding said second request to said application only if said first set matches said second set.

76. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

designating an application path of said application as restricted;

determining a destination of said operation request; and

blocking said operation request if said destination is equal to said designated application path.

77. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

compiling a list of acceptable operation requests; and

comparing said operation request to said list of acceptable operation requests.

78. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

determining a parameter value contained within said operation request; and

applying a pre-defined rule to said parameter based on said parameter type, wherein said pre-defined rule defines one or more acceptable parameter values.

79. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said operation request;

decrypting a portion of said cookie; and

modifying said cookie to reflect said decrypted portion.

80. The computer-readable medium of claim 73, wherein one of said one or more pipes comprises the steps of:

identifying a cookie associated with said reply;

encrypting a portion of said cookie; and

modifying said cookie reflect said encrypted portion.

81. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

storing said plurality of operation requests into a virtual directory;

building a dynamic range of entered values for each parameter in said plurality of operation requests;

computing an acceptable range of values for each parameter based on a statistical model applied to said dynamic range of entered values for each value;

receiving a subsequent operation request;

identifying parameter values in said subsequent operation request; and

determining if said parameter values in said subsequent operation request are within said acceptable range of values.

82. The computer-readable medium of claim 81, further comprising the steps of:

adding said parameter values in subsequent operation request to dynamic range; and

adjusting said acceptable range of values for each parameter by applying said statistical model.

83. The computer-readable medium of claim 60, wherein one of said one or more pipes comprises the steps of:

identifying a service name of said operation request;

loading a web service definition associated with said service name; and

applying a rule based on said web service definition to said operation request.

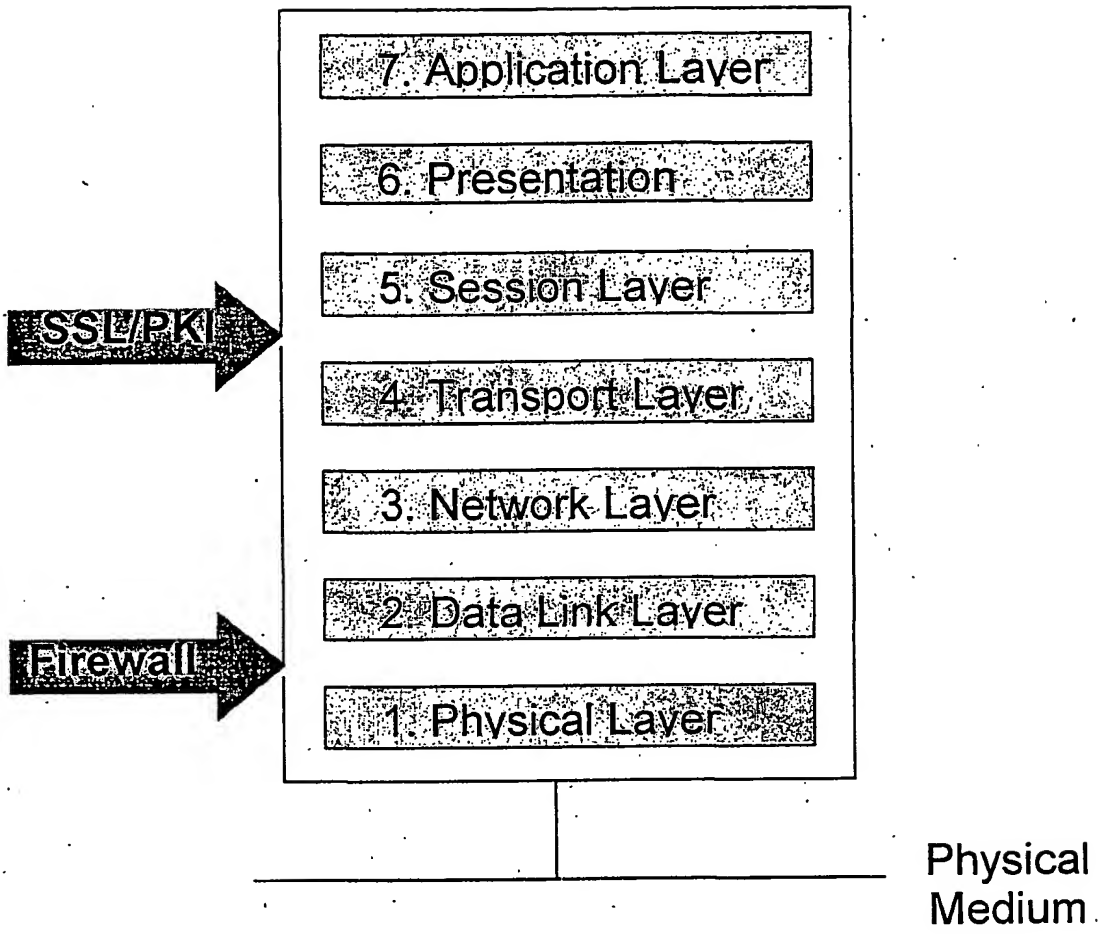


Fig. 1

(Prior Art)

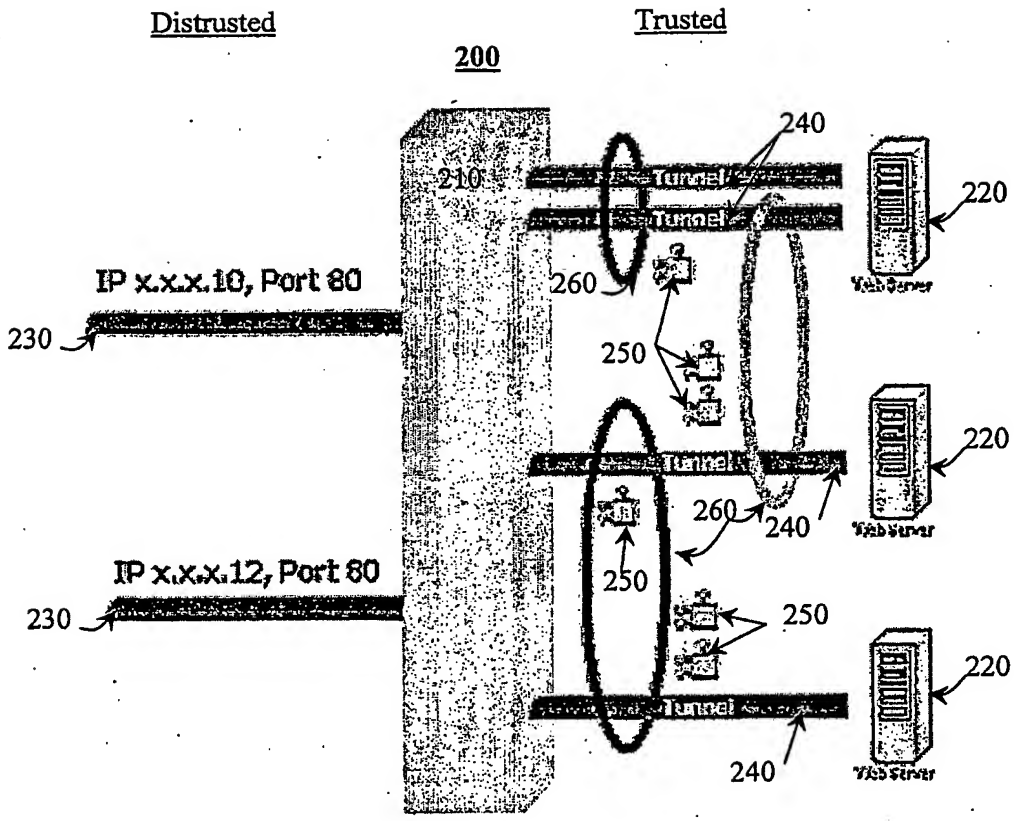

Distrusted
Trusted
200
210
240
Tunnel
Tunnel
220
Web Server
IP x.x.x.10, Port 80
230
260
250
Tunnel
220
Web Server
260
240
IP x.x.x.12, Port 80
230
250
250
Tunnel
220
Web Server
240

Fig. 2A

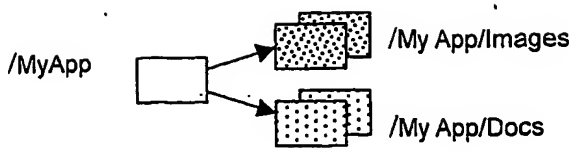


Fig. 2B

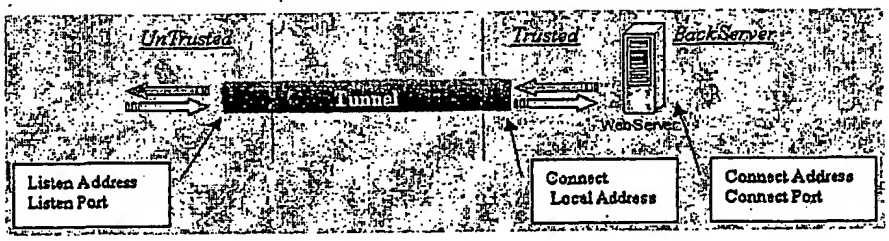


Fig. 2C

Fig. 2D

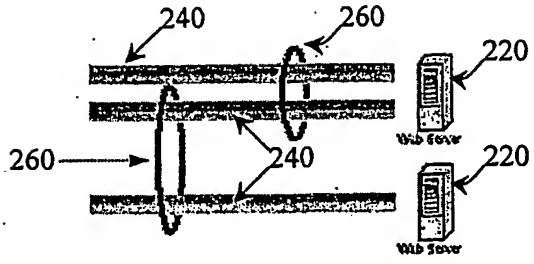


Fig. 2E

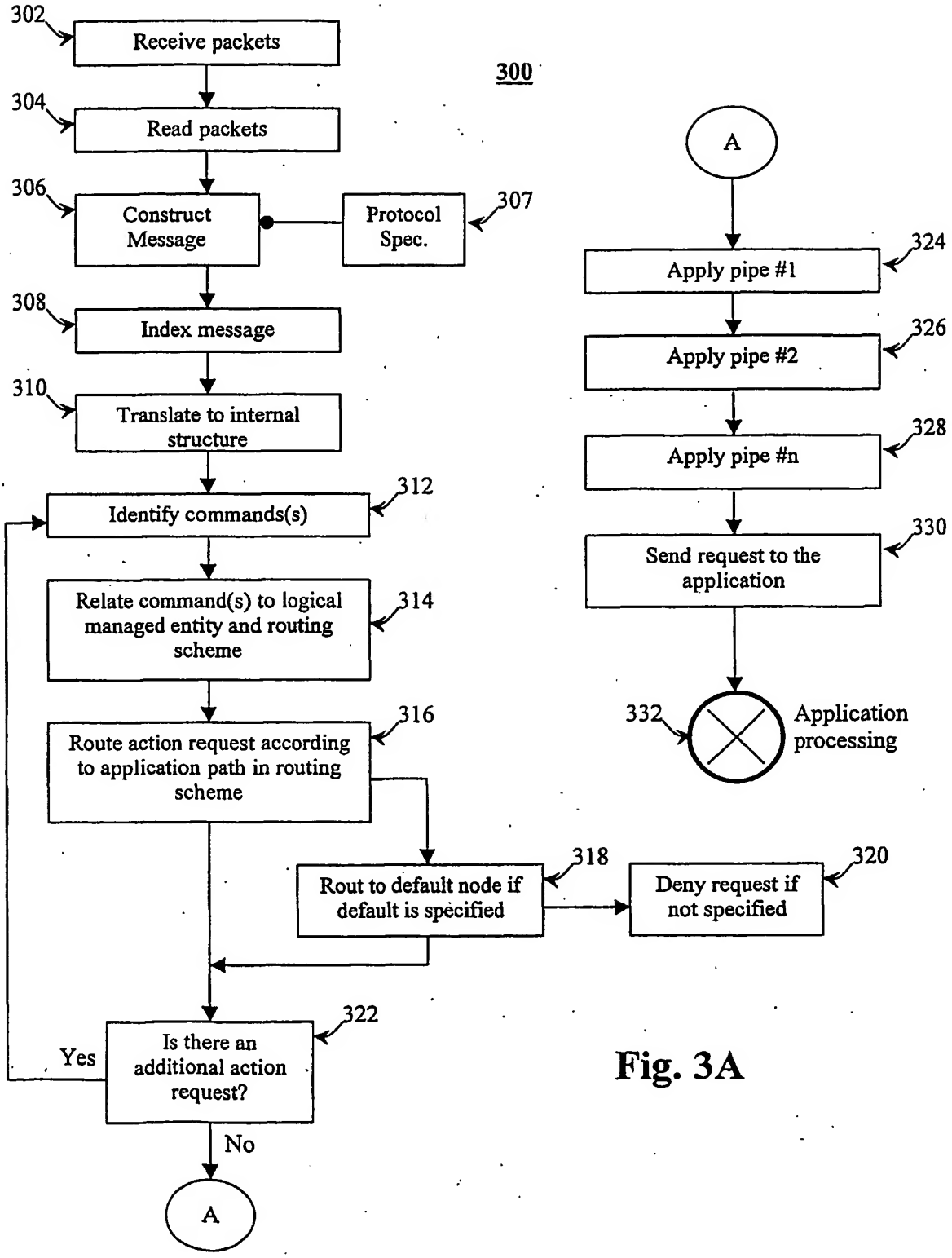

300
302
Receive packets
304
Read packets
306
Construct Message
307
Protocol Spec.
308
Index message
310
Translate to internal structure
312
Identify commands(s)
314
Relate command(s) to logical managed entity and routing scheme
316
Route action request according to application path in routing scheme
318
Rout to default node if default is specified
320
Deny request if not specified
322
Is there an additional action request?
Yes
No
A
A
324
Apply pipe #1
326
Apply pipe #2
328
Apply pipe #n
330
Send request to the application
332
Application processing

Fig. 3A

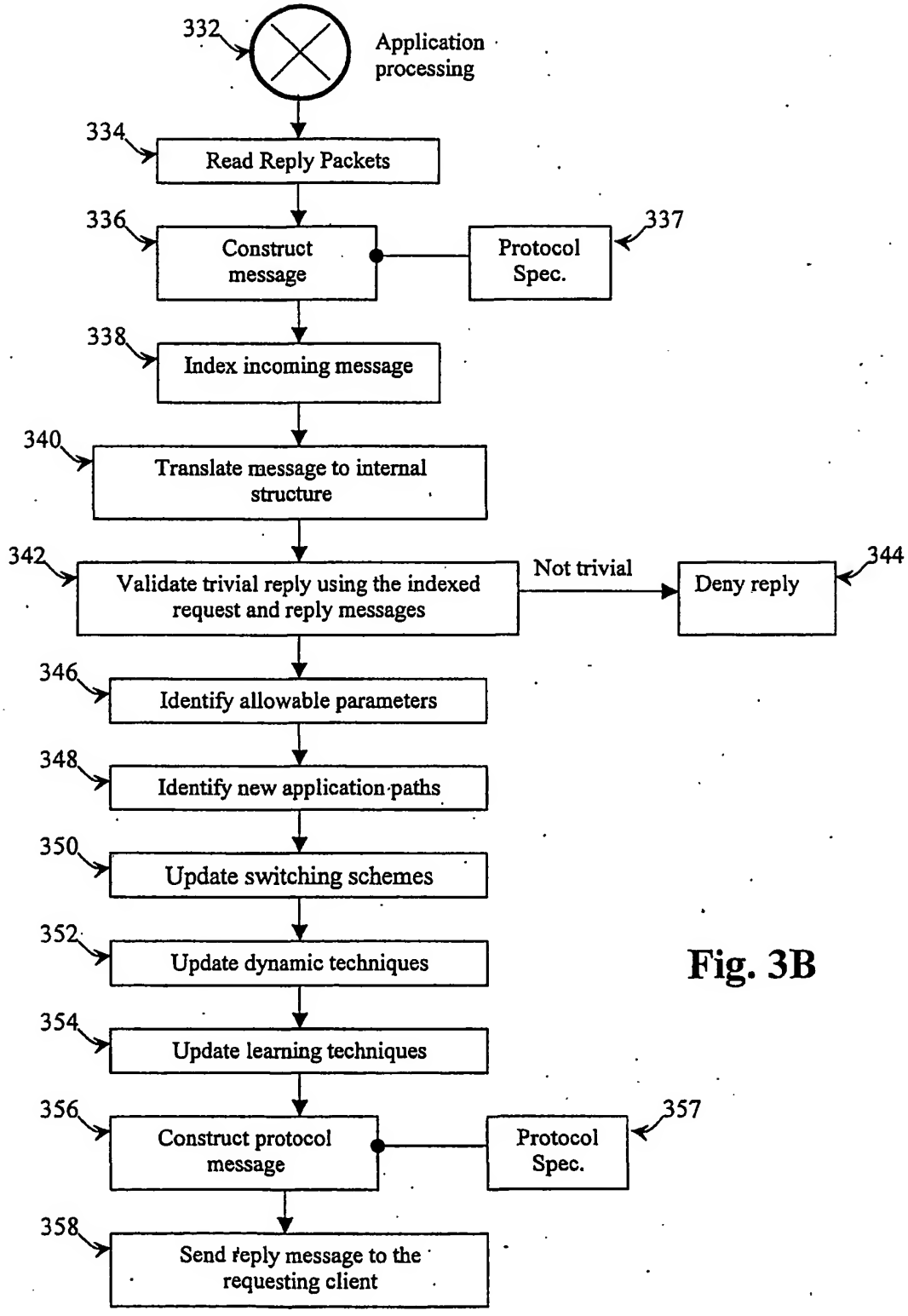

332
Application processing
334
Read Reply Packets
336
Construct message
337
Protocol Spec.
338
Index incoming message
340
Translate message to internal structure
342
Validate trivial reply using the indexed request and reply messages
Not trivial
344
Deny reply
346
Identify allowable parameters
348
Identify new application paths
350
Update switching schemes
352
Update dynamic techniques
354
Update learning techniques
356
Construct protocol message
357
Protocol Spec.
358
Send reply message to the requesting client

Fig. 3B

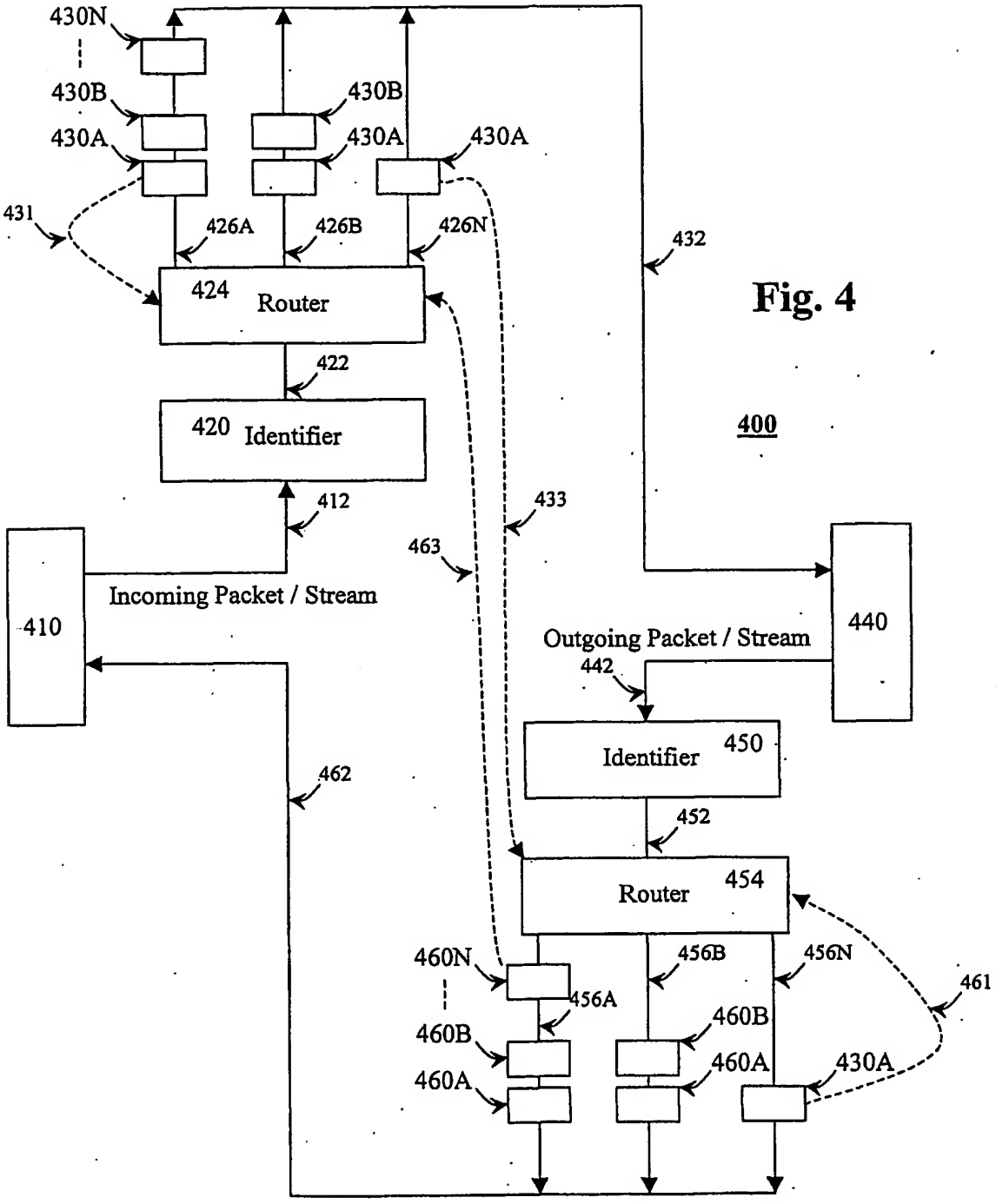

430N
430B
430A
430B
430A
430A
431
426A
426B
426N
432
424 Router
422
420 Identifier
400
412
433
463
410
Incoming Packet / Stream
440
Outgoing Packet / Stream
442
Identifier 450
462
452
Router 454
460N
456B
456N
461
456A
460B
460B
460A
430A
460A

Fig. 4

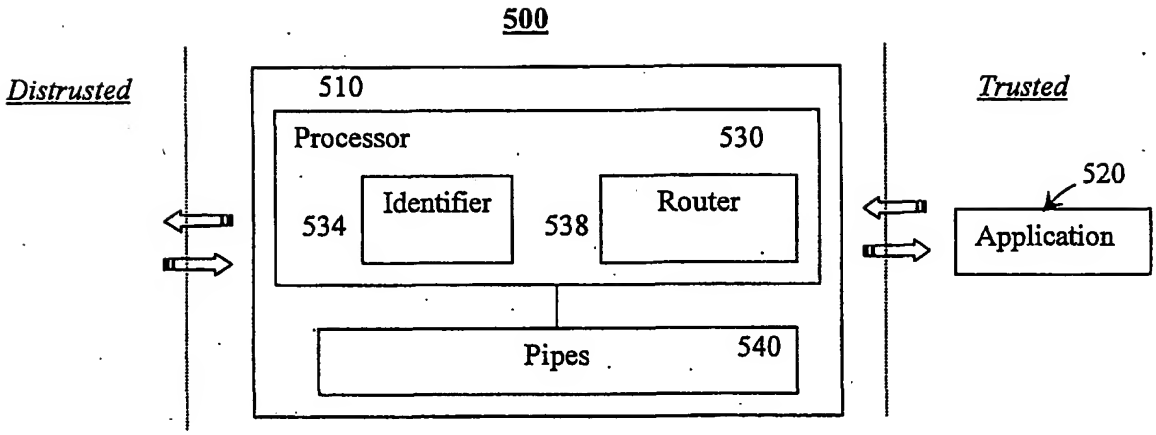

500
Distrusted
510
Processor
530
534
Identifier
538
Router
Pipes
540
Trusted
520
Application

Fig. 5

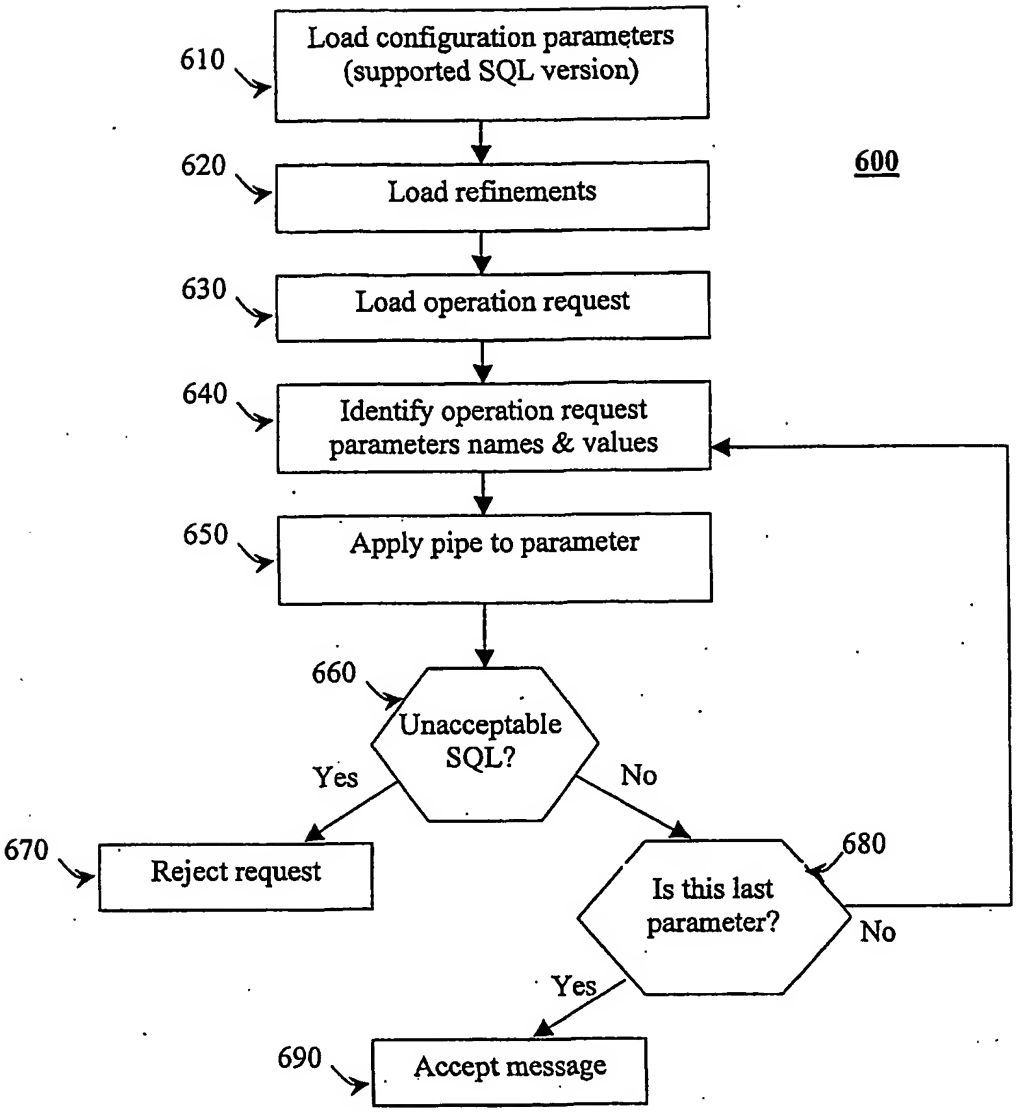

Load configuration parameters (supported SQL version)
610
Load refinements
620
600
Load operation request
630
Identify operation request parameters names & values
640
Apply pipe to parameter
650
660
Unacceptable SQL?
Yes
No
670
Reject request
680
Is this last parameter?
No
Yes
690
Accept message

Fig. 6

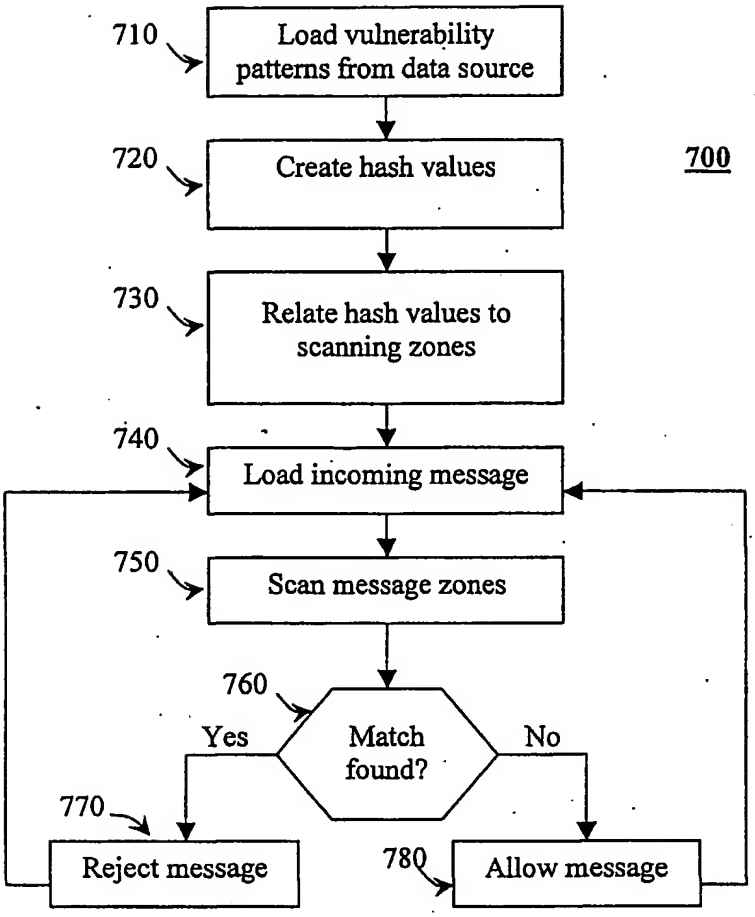

710
Load vulnerability patterns from data source
720
Create hash values
700
730
Relate hash values to scanning zones
740
Load incoming message
750
Scan message zones
760
Match found?
Yes
No
770
Reject message
780
Allow message

Fig. 7

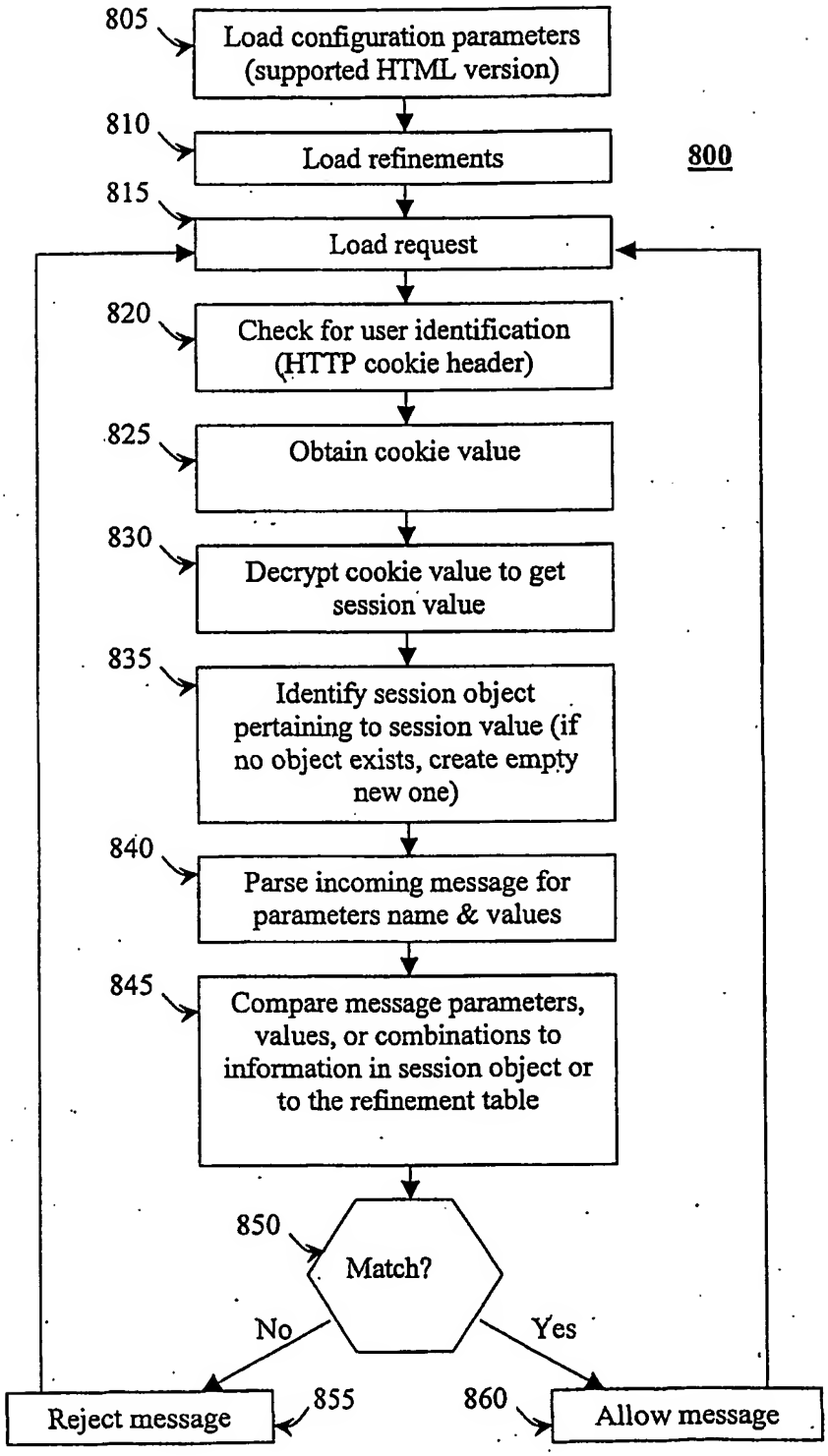

805
Load configuration parameters (supported HTML version)
810
Load refinements
800
815
Load request
820
Check for user identification (HTTP cookie header)
825
Obtain cookie value
830
Decrypt cookie value to get session value
835
Identify session object pertaining to session value (if no object exists, create empty new one)
840
Parse incoming message for parameters name & values
845
Compare message parameters, values, or combinations to information in session object or to the refinement table
850
Match?
No
Yes
855
Reject message
860
Allow message

Fig. 8

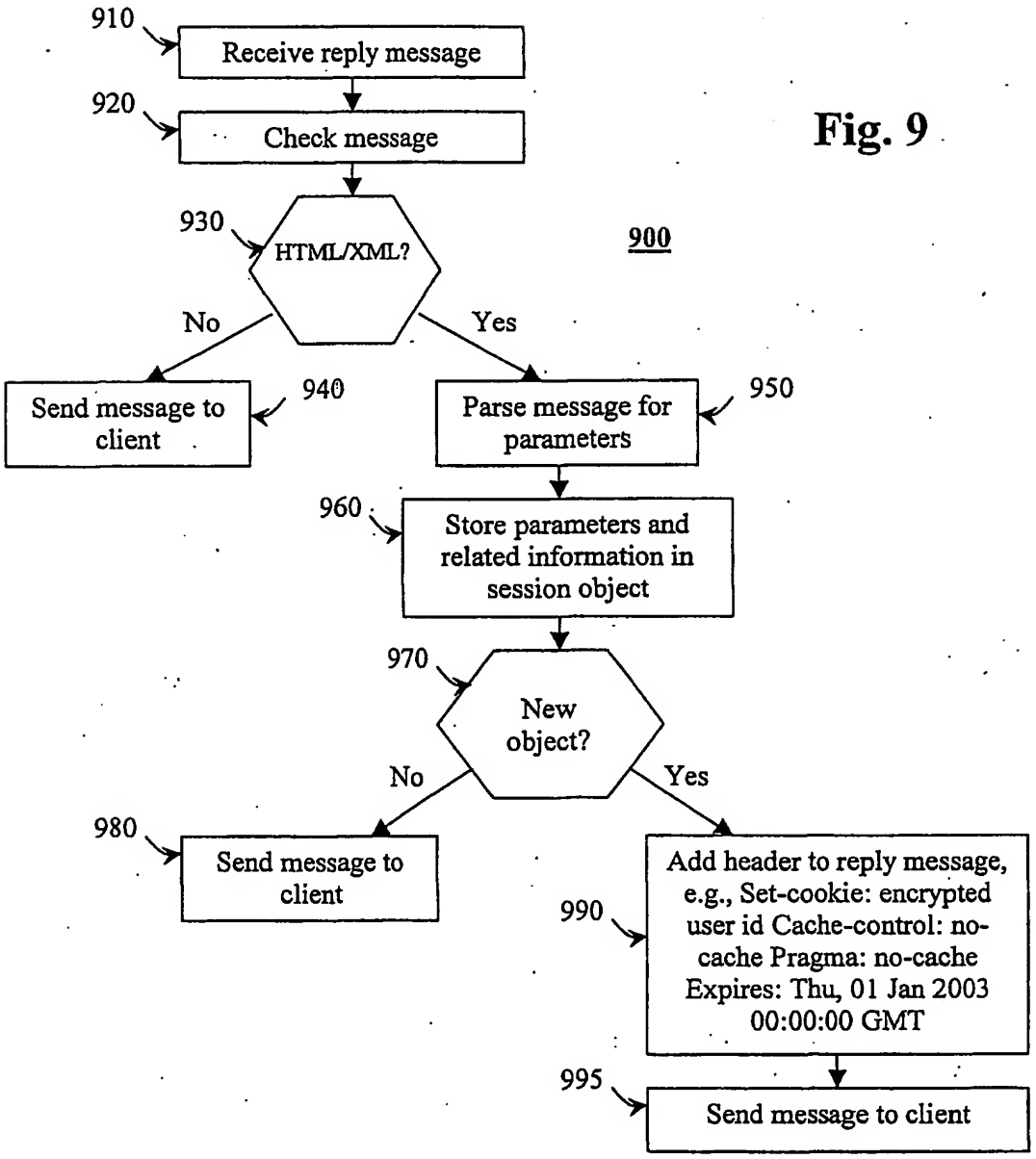

910
Receive reply message
920
Check message
930
HTML/XML?
900
No
Yes
Send message to client
940
Parse message for parameters
950
960
Store parameters and related information in session object
970
New object?
No
Yes
980
Send message to client
990
Add header to reply message, e.g., Set-cookie: encrypted user id Cache-control: no-cache Pragma: no-cache Expires: Thu, 01 Jan 2003 00:00:00 GMT
995
Send message to client

Fig. 9

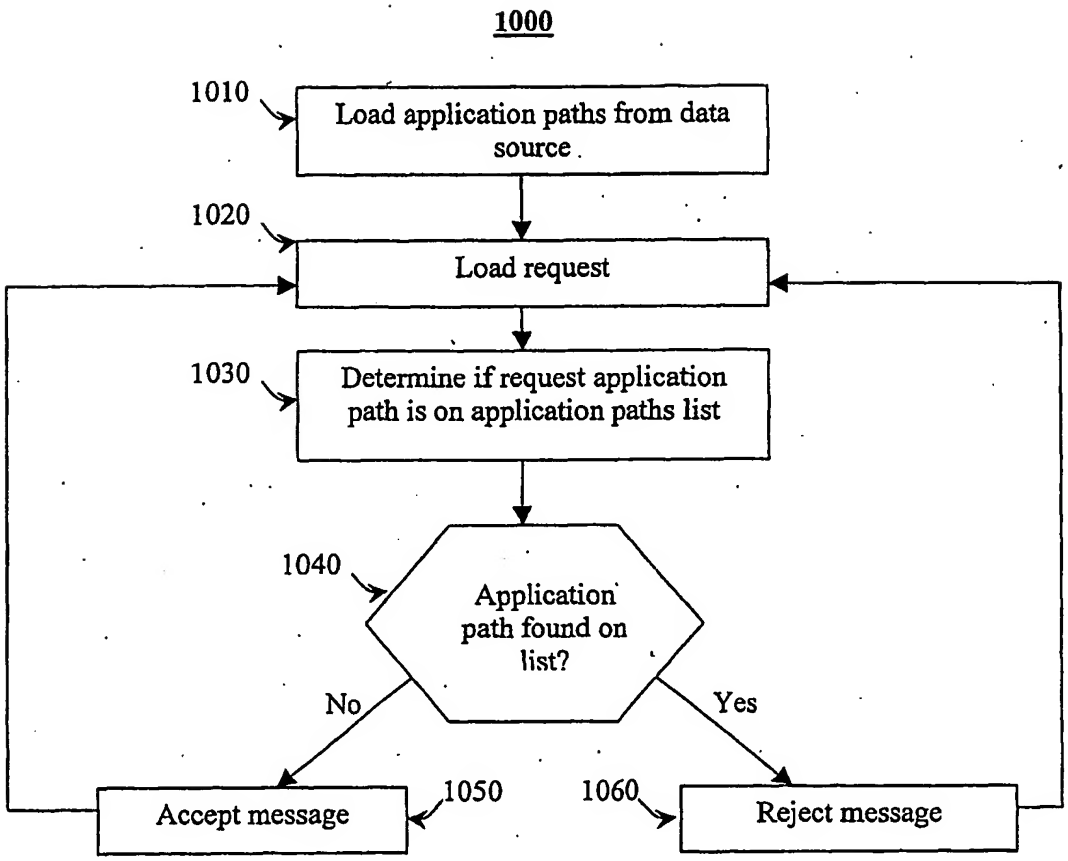

1000
1010
Load application paths from data source.
1020
Load request
1030
Determine if request application path is on application paths list
1040
Application path found on list?
No
Yes
1050
Accept message
1060
Reject message

Fig. 10

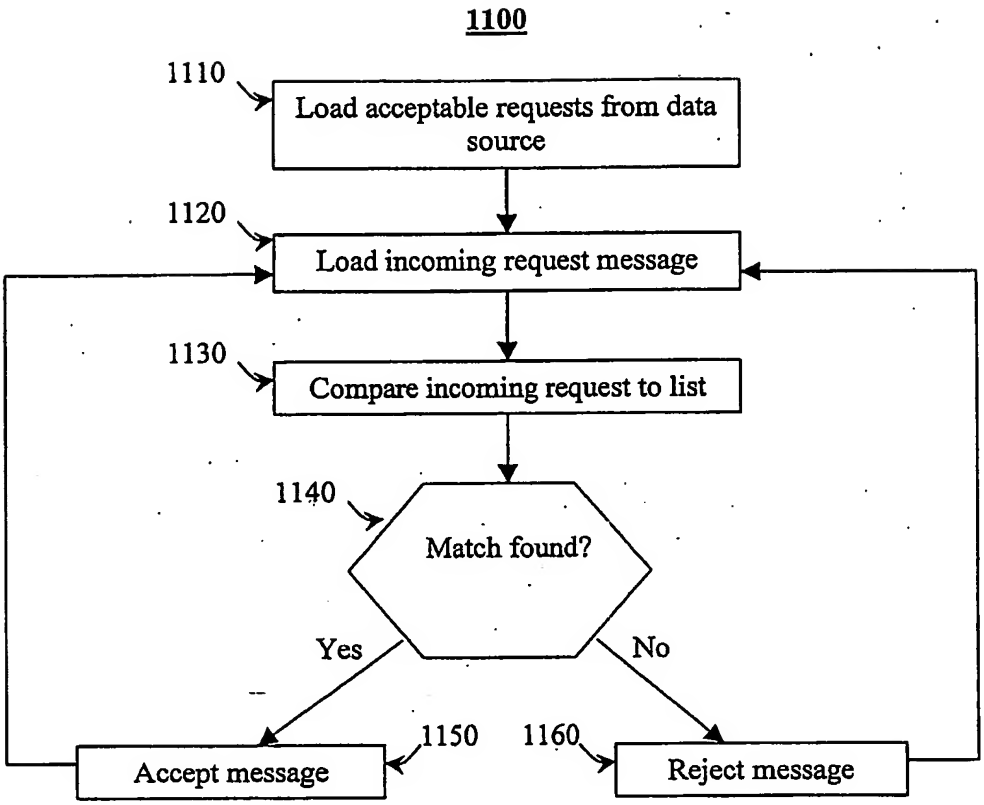

1100
1110
Load acceptable requests from data source
1120
Load incoming request message
1130
Compare incoming request to list
1140
Match found?
Yes
No
1150
Accept message
1160
Reject message

Fig. 11

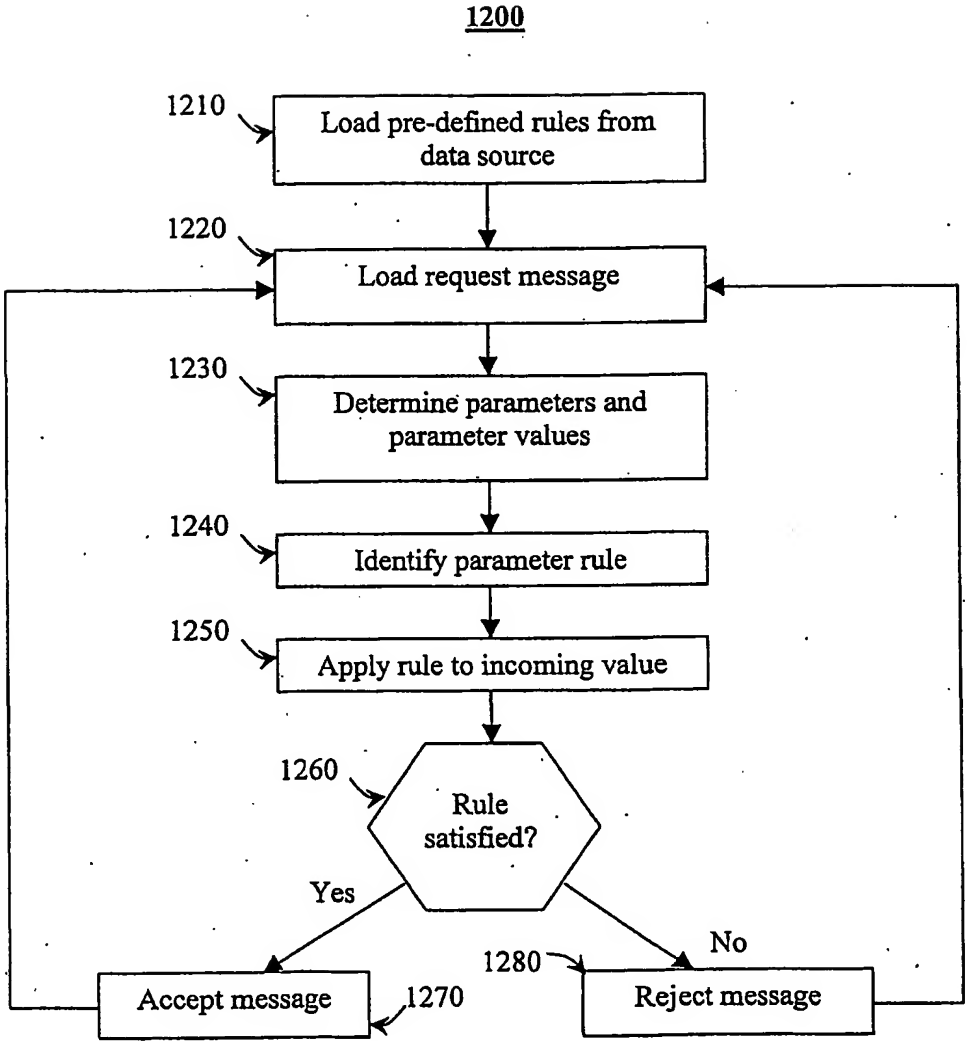

1200
1210
Load pre-defined rules from data source
1220
Load request message
1230
Determine parameters and parameter values
1240
Identify parameter rule
1250
Apply rule to incoming value
1260
Rule satisfied?
Yes
No
1280
Accept message
1270
Reject message

Fig. 12

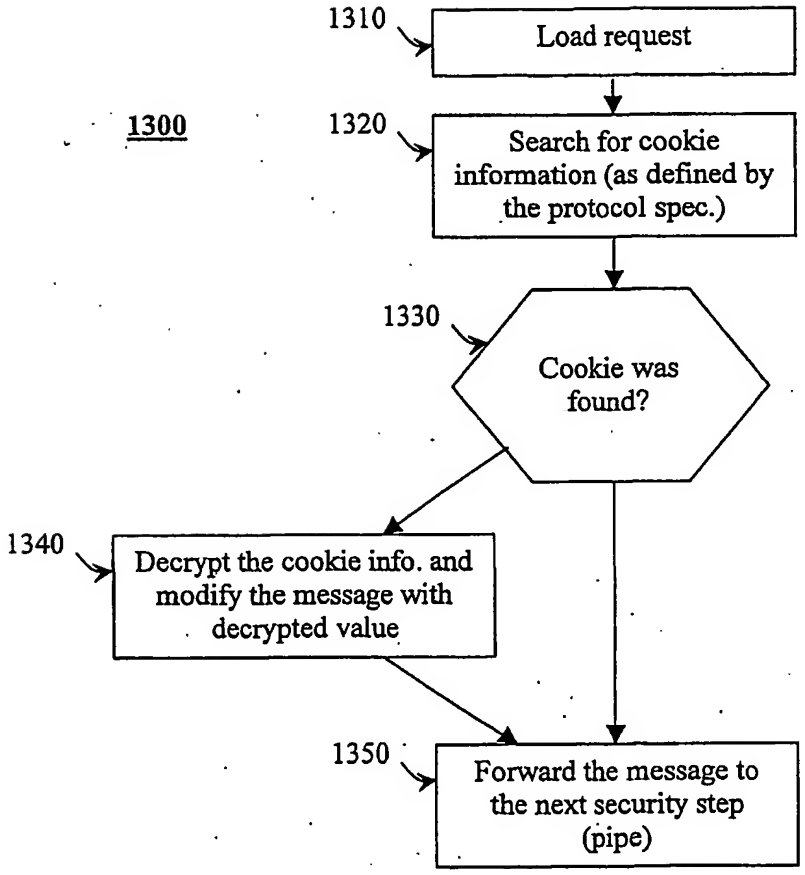

1310
Load request
1300
1320
Search for cookie information (as defined by the protocol spec.)
1330
Cookie was found?
1340
Decrypt the cookie info. and modify the message with decrypted value
1350
Forward the message to the next security step (pipe)

Fig. 13

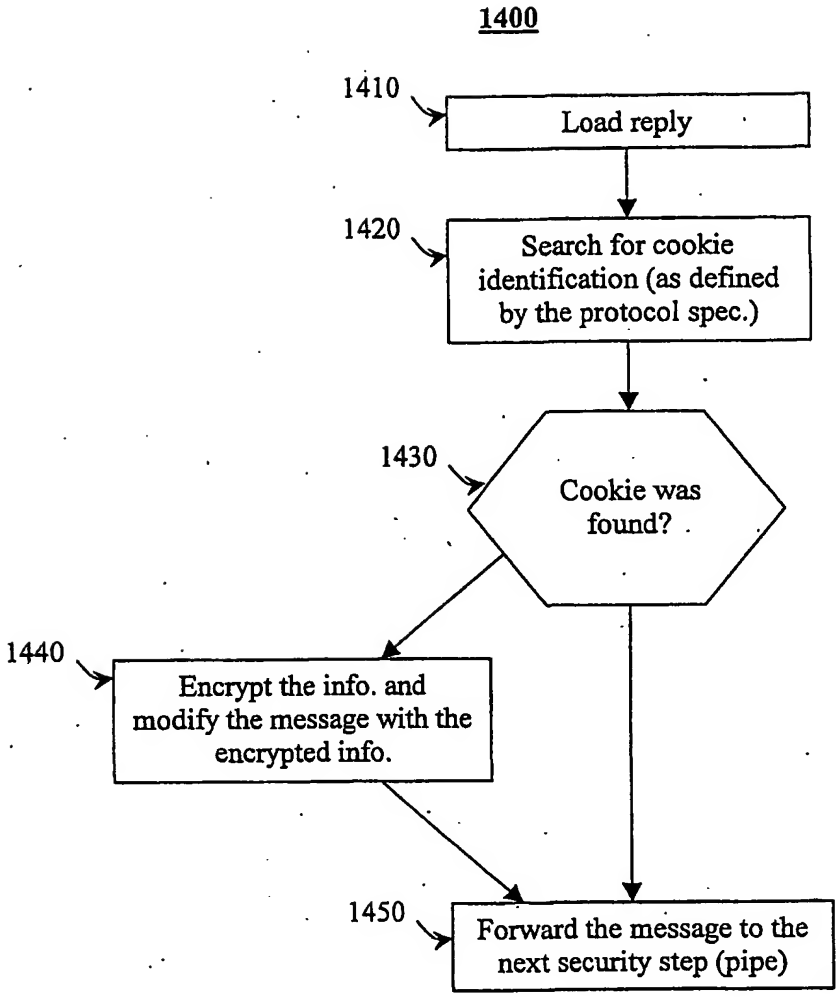

1400
1410
Load reply
1420
Search for cookie identification (as defined by the protocol spec.)
1430
Cookie was found?
1440
Encrypt the info. and modify the message with the encrypted info.
1450
Forward the message to the next security step (pipe)

Fig. 14

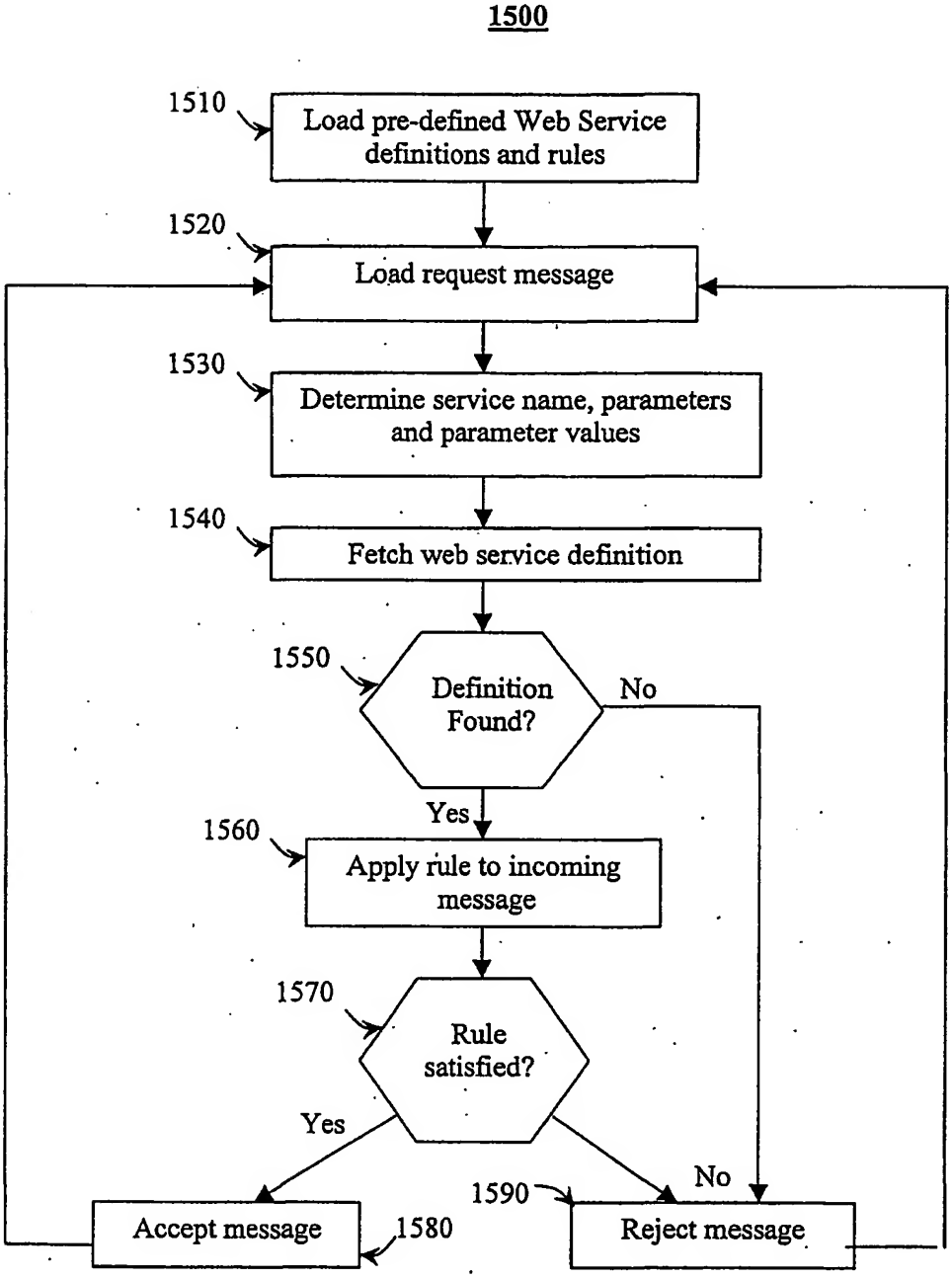

1500
1510
Load pre-defined Web Service definitions and rules
1520
Load request message
1530
Determine service name, parameters and parameter values
1540
Fetch web service definition
1550
Definition Found?
No
Yes
1560
Apply rule to incoming message
1570
Rule satisfied?
Yes
No
1590
Accept message
1580
Reject message

Fig. 15

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US02/08029

A. CLASSIFICATION OF SUBJECT MATTER
IPC(7) : GO6F 11/30
US CL : 713/201
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 713/200, 201; 709/223, 224, 229

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
N/A

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
East and Dialog; filter, gateway, proxy server, pipe, firewall, application layer, intrusion detection, web hacking,

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	APPSHIELD PROVIDES BLOCK AGAINST APPLICATION ATTACKS, BRP Publications, 07 SEPTEMBER 1999, vol. 6, issue 16.	1-4, 17-18, 27-29, 39-40, 42, 49-52, 56-57, 60, 74, 83 ——— 5-6, 11-12, 23-24, 45-46
Y	MESSMER, NEW TOOL BLOCKS WILY E-COMM HACKER TRICKS, NETWORK WORLD, 30 AUGUST 1999, pg. 15.	5-6, 11-12, 23-24, 45-46
X,P	HULME, HACKERS SNEAK THROUGH OPEN DOORS IN APPLICATIONS, INFORMATION WEEK, 2002, pg. 68.	1, 50, 60
A	US 5,623,600 A(JI et al.) 22 APRIL 1997, see col. 2, lines 39-67, col. 3, lines 51-67, col. 4, lines 56-67, col. 5, lines 39-67.	1, 50, 60

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
28 June 2002 (28.06.2002)

Date of mailing of the international search report
05 AUG 2002

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703)305-3230

Authorized officer
Gail Hayes
James R. Matthews
Telephone No. (703) 306-0426

Form PCT/ISA/210 (second sheet) (July 1998)